Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Sistema de gestión de un dron en misiones de vigilancia y reconocimiento

Estudante: Daniel García Pulpeiro
Dirección: Héctor José Pérez Iglesias
Dirección: Carlos Vázquez Regueiro

A Coruña, setembro de 2020.

A mamá, a papá y a Clara

Agradecimientos

Gracias a Carlos por su apoyo, motivación y enseñanza, incluso en momentos de vacaciones.

Agradecer en general al personal de TIC de ITG por el apoyo, y en especial a Héctor por el tiempo empleado en enseñarme magia y otras cosas de linux.

También gracias a mi familia por fomentar en mí el pensamiento crítico y permitirme experimentar desde que era pequeño.

Finalmente agradecer a los profesores de tecnología industrial, electrotecnia y física de bachillerato por enseñarme como afrontar cualquier problema, sin ellos todo esto no habría sido posible.

Resumen

El principal objetivo de este trabajo de fin de grado es desarrollar un sistema que facilite automatizar el proceso de despliegue de un dron para el desarrollo de misiones de vigilancia y reconocimiento. Se ha desarrollado un sistema que permite controlar un dron desde una interfaz gráfica accesible desde Linux, Android y Wear OS. Desde ella se pueden crear misiones definiendo la ruta del dron, así como los puntos de aterrizaje y despegue. A mayores se puede conectar un detector a la imagen en directo del dron para visualizarlo en la interfaz.

El desarrollo de este proyecto fue controlado siguiendo la metodología iterativa e incrementa. Finalmente el sistema ha sido testeado extensamente sobre un dron Anafi de Parrot. Decenas de aterrizajes autónomos exitosos en múltiples condiciones demuestran la robustez del sistema implementado. También se han realizado pruebas con distintos puntos de aterrizaje en una misma misión. Las pruebas son accesibles desde el canal de YouTube [1].

Abstract

The main objective of this final degree project is to develop a system that facilitates automating the process of deploying a drone for the usage in surveillance and reconnaissance missions. To achieve our goal, a system that allows controlling a drone from a graphical interface accessible from Linux, Android and Wear OS was developed. The creation of missions is allowed from the interface, in which you can indicate the route that the drone should take, as well as the landing and take-off points. In addition, a system was created that allows the drone image to be processed and displayed on the interface in real time.

The development of this project was controlled following the iterative and incremental methodology. Finally the system has been extensively tested on a Parrot Anafi drone. Dozens of successful autonomous landings in multiple conditions demonstrate the robustness of the implemented system. Tests have also been carried out with different landing points in the same mission. Tests are accesible from the YouTube channel [1].

Palabras clave:

- Dron
- Aterrizaje automático
- Estación de control terrestre
- Vuelo autónomo

Keywords:

- Drone
- Autonomous landing
- Ground control station
- Autonomous flight

Índice general

Índice de figuras

Índice de tablas

Lista de Algoritmos

Capítulo 1

Introducción

En este capítulo se explicarán la motivación del presente proyecto, sus objetivos concretos, y la estructura de la memoria.

1.1 Motivación

Actualmente los drones están siendo usados en multitud de escenarios para la ejecución de diversas tareas, por ello consideramos interesante la creación de un sistema que permita realizar automáticamente alguna de ellas durante períodos largos de tiempo, sobre todo las que son repetitivas o engorrosas, y por lo tanto con más probabilidad de error.

La idea consiste en simplificar y agilizar el despliegue de un dron y la mejora de la gestión de los recursos disponibles. Especialmente la automatización del proceso de aterrizaje, y la gestión del tiempo de vuelo de un dron (actualmente limitado a unos minutos) y de recarga o sustitución de las baterías,

1.2 Objetivos

1. Diseñar un sistema capaz de crear misiones durante largos períodos de tiempo. Cada misión constará como mínimo de seis fases: el despegue, aterrizaje, recarga (simulando disponer de un sistema automático de recarga, la ruta para ejecutar la tarea (propuesta por un operador o por un algoritmo externo), y los desplazamientos desde el punto inicial y final de esa ruta.

2. Diseñar un sistema de control de un dron que sea capaz de ejecutar las distintas fases de las misiones. Básicamente la navegación mediante posición geolocalizada usando el Global Navigation Satellite System. (GNSS) y, sobre todo, aterrizaje (fase crítica en todo el proceso).

3. Diseñar un mecanismo que identifique el punto de aterrizaje en diferentes condiciones de iluminación y de visibilidad (distinta incidencia del sol, sombras, baja visibilidad, etc). Debe ser robusto a oclusiones parciales (p.e. caída de hojas o otros objetos en el lugar de aterrizaje.

4. Demostración y pruebas del sistema en un simulador 3D, y en la medida de los recursos disponibles, también en condiciones reales.

1.3 Estructura de la memoria

En los siguientes capítulos de la memoria se explicará en más detalle el desarrollo llevado a cabo en este proyecto. A continuación se explican de forma breve cada uno de dichos capítulos:

- Introducción: es el presente capítulo, con la motivación, objetivos y la estructura de la memoria.

- Trabajo relacionado y fundamentos: donde se analizan los trabajos más relacionados con el proyectos, se describen las tecnologías empleadas y se indican los fundamentos teóricos más importantes.

- Gestión del proyecto: en este capítulo se analizan los requisitos del sistema, se establece la metodología de desarrollo, su ejecución y los costes incurridos.

- Diseño: en este apartado se explica la arquitectura general del sistema, así como la interfaz propuesta y el modelo de datos propuesto.

- Implementación y pruebas: en este capítulo de explican los detalles de implementación más relevantes y se documenta el extenso conjunto de pruebas experimentales realizado.

- Solución desarrollada: en este apartado se muestran las diferentes interfaces gráficas de usuarios desarrollado para tres sistemas operativos diferentes: Linux, Android y Wear OS.

- Conclusiones y trabajo futuro: En este último capítulo de la memoria se habla de las conclusiones obtenidas a lo largo del desarrollo, así como de futuras mejoras y cambios en el proyecto

Capítulo 2

Trabajo relacionado y fundamentos

Este capítulo comenzará mostrando el trabajo relacionado, continuará con las tecnologías utilizadas y finalizará con un repaso de los ejes de una aeronave.

2.1 Trabajo relacionado

En esta sección se mostrarán algunas de las soluciones que hay en el mercado para solventar un problema similar al que se plantea.

2.1.1 PaparazziUAV

PaparazziUAV [2]: Proyecto de software y hardware de drones de código abierto que abarca sistemas de piloto automático y software de estaciones terrestres para multicópteros / multirrotores, ala fija, helicópteros y aviones híbridos que se fundó en 2003. Está enfocado al vuelo autónomo.

Paparazzi se asemeja a este proyecto en su misión de intentar facilitar el vuelo autónomo de drones, pero difiere en el enfoque llevado a cabo. En este proyecto se hizo una interfaz simple capaz de ser gestionada por una persona sin apenas conocimientos de como funciona un dron, mientras que en el caso de Paparazzi las capacidades para especificar todo tipo de parámetros son mucho mayores. En la figura 2.1 se puede ver una imagen de la interfaz de este software.

2.1.2 QGROUNDCONTROL

QGROUNDCONTROL [3] proporciona control de vuelo completo y planificación de misiones para cualquier dron habilitado para Micro Air Vehicle Link (MAVLink). Tiene como objetivo principal facilitar el control de drones a usuarios y desarrolladores profesionales. [1]

[1]Micro Air Vehicle Link es un protocolo para comunicarse con vehículos pequeños no tripulados

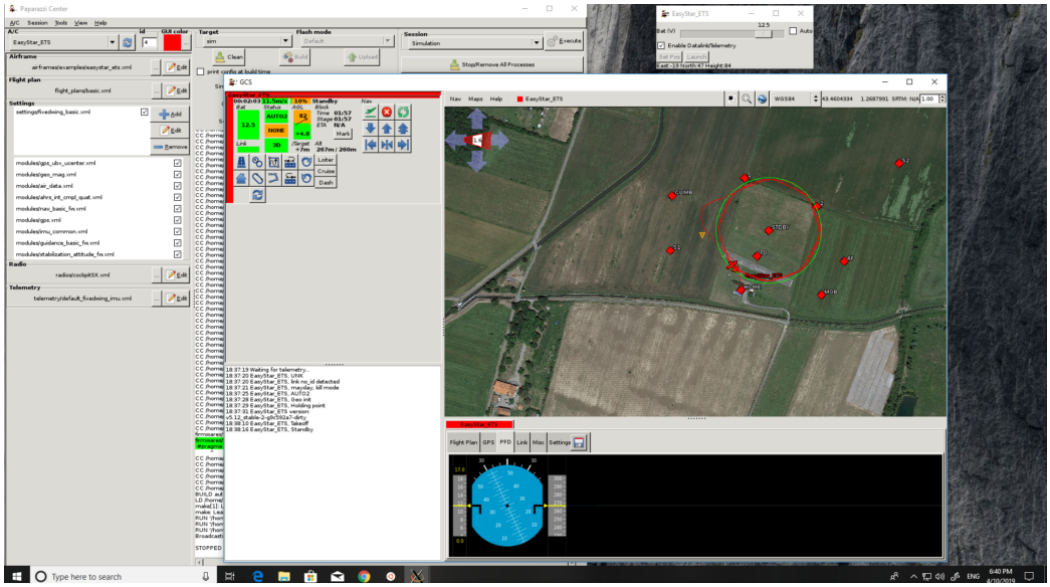


Figura 2.1: Imagen del software PaparazziUAV.

QGroundControl dispone de una interfaz limpia, más parecida da la desarrollada en este proyecto que en el caso de Paparazzi, pero tampoco implementa la función de aterrizaje de precisión, ni está disponible en versión smartphone. Se puede ver una imagen de esta interfaz en la figura 2.2).

2.1.3 SKEYETECH

SKEYETECH [4] es un sistema 100% autónomo, pensado para vigilancia de áreas sensibles. Dispone de funciones de aterrizaje automático y control de misiones además de una una caja para el despliegue y recarga del dron.

Skeyetech sería un competidor directo que ya dispone de un sistema de recarga de drons que en este proyecto no se ha desarrollado. Está pensada como una solución profesional en la que los posibles detectores de personas o intrusos no está integrada en la misma. En la web no se muestra ningún tipo de interfaz software para el control del dron, por lo que carecemos de información para realizar una comparación en ese aspecto. En la figur 2.3 se puede ver como es la caja de esta plataforma.

2.1.4 Parrot-Hoverseen

La unión entre las empresas Parrot y Hoverseen [5] ha producido el diseño de una caja con la que proporcionar aterrizaje de precisión y recarga de drons Parrot Anafi. Se puede ver una imagen de este producto en la figura 2.4.

Esta solución proporciona la caja de aterrizaje y recarga para una versión modificada del

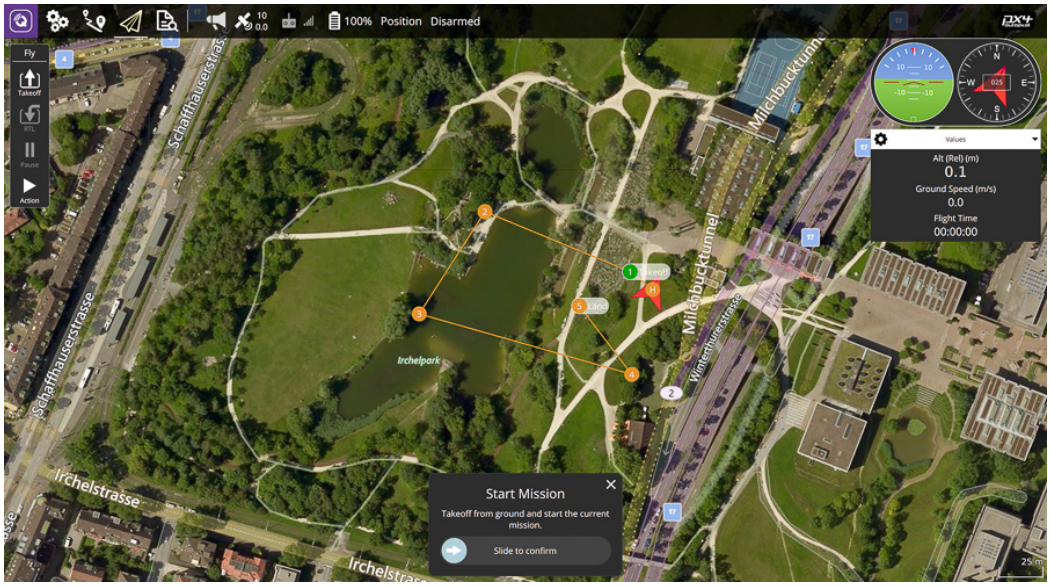


Figura 2.2: Imagen del software QGROUNDCONTROL.

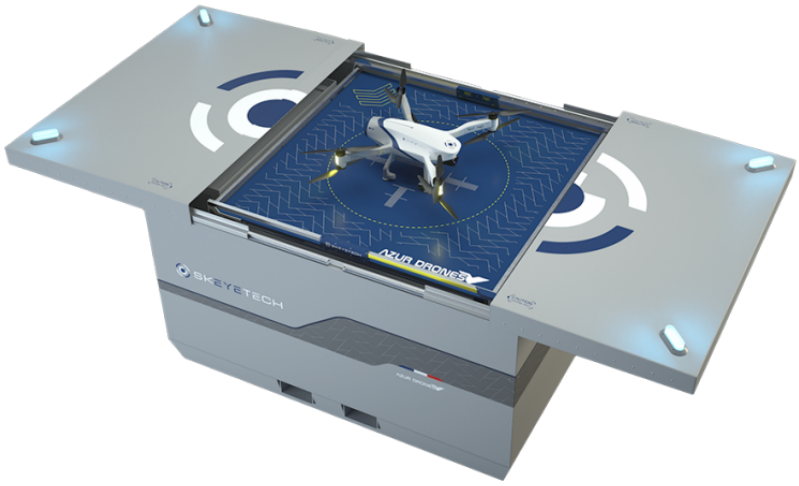


Figura 2.3: Imagen de la plataforma SKEYETECH.

Parrot Anafi, no se dispone de información de ningún tipo de interfaz gráfica que un usuario final pueda usar para controlar la zona que deba vigilar el dron.

2.2 Tecnologías utilizadas

A continuación se mencionan las principales tecnologías utilizadas, están organizadas en entorno de desarrollo, lenguajes de programación, librerías utilizadas, base de datos, interfaz, Parrot Anafi y otros.

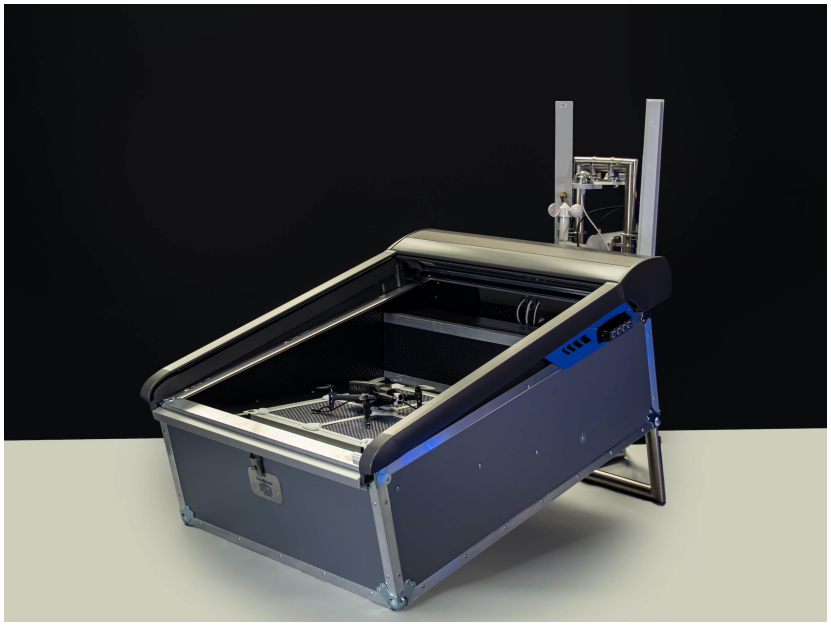


Figura 2.4: Imagen del producto de Parrot-Hoverseen.

2.2.1 Entorno de desarrollo

- **Debian 10** [6]: Sistema operativo de software libre que testea profundamente los paquetes antes de publicarlos en el repositorio estable. Es la base de muchas otras distribuciones GNU/Linux.

Logo de Debian.

- **DWM** [7]: Gestor de ventanas minimalista e ultraligero de código libre. Permite realizar todo tipo de modificaciones, teniendo que hacerse estas directamente en el código fuente escrito en C. Usar este gestor de ventanas nos permite trabajar de forma mucho mas ágil que en caso de hacer el proyecto con un entorno de escritorio más pesado como Kde o Gnome. Su uso de RAM o VRAM es mínimo, lo que nos permitirá utilizar una configuración para darknet más agresiva cuando entrenemos You only look once. (YOLO).

- **NeoVim** [8]: Fork de vim [9] que facilita añadir y crear nuevos plugins. Nos facilitará

realizar modificaciones en el código de python cuando estemos probando el sistema.

Logo de NeoVim.

- **Tmux** [10]: Multiplexor de terminal. Se usará para tener varias terminales en una misma ventana además de porque en caso de cerrar la ventana de la terminal por equivocación no se pararán los procesos lanzados, pudiendo volver conectarnos a la sesión de tmux que contiene dichos procesos desde otro terminal nuevo. Esta característica nos sería de gran ayuda para no perder sin querer la conexión con el dron cuando la reconexión al mismo todavía no estaba implementada.

- **VSCode** [11]: Editor de código fuente creado por Microsoft. Será usado principalmente para desarrollar la interfaz en Flutter.

Logo de VSCode.

- **Parrot Sphinx** [12] Herramienta de simulación creada por parrot para facilitar el desarrollo de software para sus drones. Parrot sphinx permite correr el firmware de los drones en un entorno de simulación basado en Gazebo.

Logo de Parrot Sphinx.

2.2.2 Lenguajes de programación

- **Python3**[13]: Lenguaje interpretado de propósito general. Utilizado en el sistema de control.

Logo de Python.

- **C++**[14]: Lenguaje compilado de propósito general. Creado como una extensión a "C" con clases, proporciona una legibilidad y rendimiento como pocos otros lenguajes. Es el lenguaje usado en el detector.

Logo de C++.

- **Dart**[15]: Lenguaje creado por Google para facilitar el desarrollo rápido de interfaces en todas las plataformas. Flutter 2.2.5 está creado sobre este lenguaje.

Logo de Dart.

2.2.3 Librerías utilizadas

- **Aruco**[16]. Librería con marcadores basada en OpenCV.
- **opencv_python**. Paquete con OpenCV para python.
- **Olimpe** [17]. Librería desarrolada por Parrot para el control de sus drons.

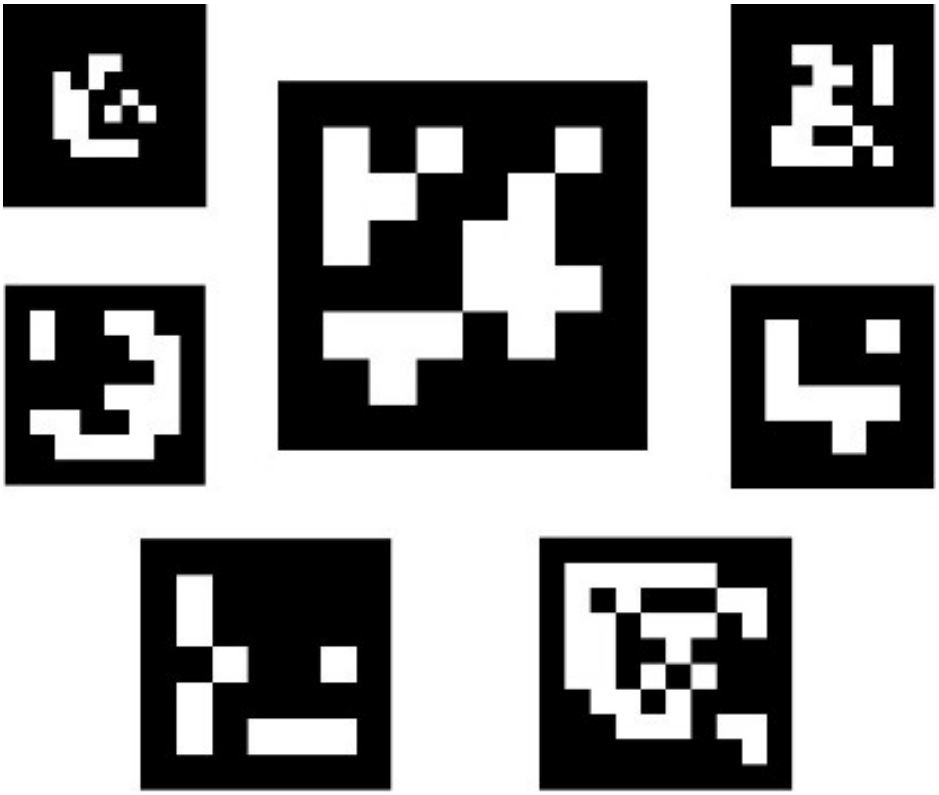


Figura 2.5: Ejemplos de varios marcadores visuales Aruco.

- **Flask** [18]. Paquete de Python para el desarrollo de APIs.
- **Paho-mqtt** [18]. Paquete de Python para la comunicación con el Broker MQTT.
- **WebsocketServer** [19]. Paquete de Python para el para crear servidores websocket.
- **Psycopg2** [20]. Paquete de Python para la conexión con la base de datos PostgreSQL.
- **flutter_map** [21]. Implementación de Leafleft en dart para Flutter.
- **flutter_bloc**[22]. Paquete de para facilitar el manejo de estado.
- **Pygame** [18]. Paquete de Python para el desarrollo de juegos en 2D. Será usado para controlar el dron desde el ordenador en las primeras iteraciones del proyecto.

Logo de Pygame.

- **OpenCV** [23] es una librería de software de visión artificial y aprendizaje automático de código abierto. Se creó para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Está compartido bajo la licencia BSD, lo que facilia e uso y desarrollo de la librería por parte de las empresas.

Logo de OpenCV.

2.2.4 Sistema de gestión de base de datos

- **PostgreSQL** [24]. Sistema de gestión de base de datos de código abierto.

Logo de PostgreSQL.

- **PostGIS** [25]. Extensión de PostgreSQL para la gestión de datos espaciales. En el proyecto que tratamos, Postgis se utiliza tanto para almacenar las coordenadas de los puntos por lo que debe pasar el dron cuando realice una misión, como para guardar información de los metadatos de vuelo, entre los cuales se encuentra la posición del dron.

Logo de Postgis.

2.2.5 Interfaz

La interfaz de usuario ha sido desarrollada con Flutter.

Flutter es un *framework* multiplataforma escrito en Dart [15] creado por Google para el desarrollo de interfaces gráficas. Permite compartir una amplia parte del código escrito para su uso en diferentes plataformas, Android, IOS, Web y escritorio.

Logo de Flutter.

2.2.6 Otros

- **Mosquitto**[26]: Broker MQTT usado para la redistribución de metadatos a las diferentes interfaces.
- **Docker** es un proyecto de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. En nuestro caso, la base de datos PostgresSQL y la extensión Postgis están integradas dentro de un contenedor docker.

Logo de Docker.

- **QGIS** Sistema de información geográfico que entre muchas otras características, permite conectarse con PostgreSQL y Postgis. De gran utilidad en etapas tempranas del proyecto para visualizar en un mapa la información geográfica guardada en la base de datos.

Logo de QGIS.

- **Microsoft Planner**: Utilizada para visualizar la planificación.

Logo de Microsoft Planner.

- **Draw.io**: Para la creación de los diagramas mostrados en Capítulo 4.

Logo de Draw.io.

2.2.7 Parrot Anafi

Tras hacer una búsqueda de los posibles drones para llevar este proyecto a cabo, la facilidad que proporciona Parrot al desarrollador para la creación de sistemas que integren sus drones hicieron decantar la balanza por esta marca.

Figura 2.6: Imagen del dron Parrot Anafi empleado en este proyecto.

Entre los diferentes modelos de los que disponen destacan el Parrot Bebop2 y el Parrot Anafi (figura 2.6), siendo la última opción la elegida. A continuación destacamos las principales características por las cuáles lo hemos elegido para este proyecto:

- **Viento**: soporta viento de hasta 50 Km/h.
- **Batería**: 26 minutos de vuelo, lo que nos permitirá realizar pruebas relativamente largas. Disponemos de dos baterías, que al ser fáciles de cambiar, nos permite realizar ciclos de pruebas más largos.
- **Cámara**: 4K con HDR y **Gimbal**. Debido al método de aterrizaje que teníamos pensado desarrollar, consideramos de especial importancia que el dron dispusiera de gimbal con control electrónico para direccionarlo, al menos, en el eje vertical.
- **Rango de vuelo**: Entorno a 4 Km, aunque en la realidad no debemos llegar a esa distancia, ni tampoco sobrepasar los 120 m de altura.
- **Olympe SDK**: Compatible con el Software Development Kit. (SDK) de parrot para drones.

Además de las características anteriormente listadas, Parrot dispone de un modelo de Anafi que integra una cámara térmica que valdría para la creación de un detector de personas basado en este tipo de imágenes, y que se podría usar para la búsqueda de personas desaparecidas o cualquier otro foco de calor (p.e. incendios).

2.3 Fundamentos teóricos

2.3.1 Ejes de rotación

El movimiento de un dron puede ser controlado en los tres ejes de rotación y en la vertical. En la figura Figura 2.7 se puede ver más claramente.

- **Eje X**: Representa la dirección hacia delante o hacia atrás del dron. Para controlar un desplazamiento en esta dirección debemos crear una rotación sobre el eje Y del dron. A esta rotación se le denomina **pitch**.
- **Eje Y**: Representa el movimiento lateral hacia la derecha o a la izquierda. Para controlar un desplazamiento en esta dirección debemos crear una rotación sobre el eje X del dron. A esta rotación se le denomina **roll**.
- **Eje Z**: Representa el movimiento sobre la vertical. A una rotación sobre el eje Z se le denomina **yaw**.

Para conseguir mover el dron en la vertical basta con aplizar la misma fuerza en todos los motores. En el SDK de Parrot [17] a esta fuerza se le conoce como “gaz”.

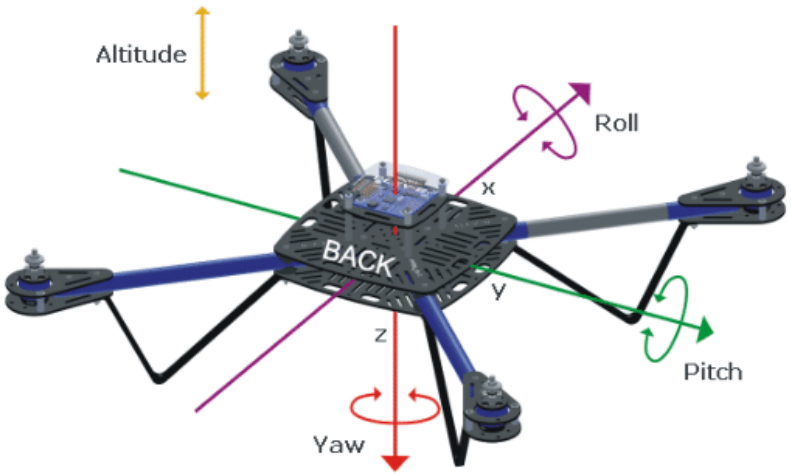


Figura 2.7: Ejes de giro de un dron.

Capítulo 3

Gestión del proyecto

El presente capítulo se comenzará explicando cuáles han sido los requisitos del proyecto, seguirá con la metodología usada para el desarrollo de este producto, continuará con la planificación y finalizará con una evaluación de los costes del mismo.

3.1 Análisis de requisitos

En esta sección se van a explicar los requisitos que debe cumplir nuestro sistema para poder considerarlo acabado. Comenzaremos definiendo cuales son los requisitos funcionales y terminaremos por los no funcionales.

3.1.1 Requisitos funcionales

Los requisitos funcionales definen los patrones de comportamiento el sistema ante una determinada entrada:

RF1 El sistema debe ser capaz de detectar la plataforma de aterrizaje en condiciones de luz diurna. Requisito inicial.

RF2 El sistema debe ser capaz de aterrizar el dron a una distancia máxima de 30cm del centro de la plataforma. La distancia será tomada desde el centro de la plataforma de detección hasta la cámara del dron. Esto permitiría la creación de una base de recarga de menos 1m x 1m para el dron. Requisito inicial.

RF3 Debe haber una interfaz gráfica que permita el control del dron. Requisito inicial.

RF4 Se debe poder acceder de forma remota al video y metadatos del dron. Este requisito surge en en seguimiento de la iteración 2.

RF5 Desde la interfaz se deberán poder crear misiones. Se podrá especificar para cada misión las coordenadas Global Positioning System. (GPS) del punto de despegue y aterrizaje y

los puntos por los que debe pasar el dron, además de la altura a la que están. Requisito inicial.

RF6 Desde la interfaz gráfica será posible visualizar la imágenes en tiempo real del dron y del detector, así como la posición del dron en un mapa. Este requisito surge en en seguimiento de la iteración 3.

RF7 Desde la interfaz gráfica de linux se debe poder controlar el dron por completo, lo cual implica poder ejecutar una misión, parar dicha misión y mover el dron con las teclas WASD y las flechas del teclado. Este requisito surge en en seguimiento de la iteración 5.

RF8 En la pantalla de control se debe poder sustituir la vista principal por cualquiera de las secundarias. Se define como vista principal a la que se ve en grande, y secundarias a las dos restantes. Este requisito surge en en seguimiento de la iteración 6 7.

RF9 El sistema debe ser capaz de realizar una misión en momentos programados. Requisito inicial.

RF10 Desde la vista principal se deberá poder ver información de la altura, calidad de la conexión, y batería del dron. Este requisito surge en en seguimiento de la iteración 5 6.

RF11 El sistema de control deberá implementar un modo que permita volver a conectarse al dron en caso de error. Requisito inicial.

RF12 Deberá haber un sistema que determine si es seguro ejecutar una misión en base a la distancia total de la misma, la posición del dron y la batería restante. Requisito inicial.

RF13 Deberá existir un modo que calcule una ruta en un área. Requisito inicial eliminado en el seguimiento de la iteración 7 para dar prioridad a **RF15**.

RF14 En el mapa de la pantalla de control ser debe ver superpuesta la ruta que está realizando el dron.

RF15 El dron debe poder despegar y aterrizar desde varios puntos en una misma misión. Este requisito surge en en seguimiento de la iteración 7.

3.1.2 Requisitos no funcionales

El apartado de requisitos no funcionales de un análisis de requisitos, también conocido como atributos de calidad, define aquellas necesidades que no se corresponden con una manera de proceder del sistema ante una entrada, sino que describen una característica de funcionamiento,

RNF1 La latencia entre que una imagen es capturada en el dron, y esa misma imagen puede ser visualizada en el sistema de control, no deberá superar los 200 milisegundos y se tratará de optimizar todo lo máximo posible de cara a reducir dicha latencia. Este valor es crítico en el momento de aterrizaje.

RNF2 La latencia en la imagen sin procesar mostrada en la interfaz gráfica deberá ser inferior a 500 ms para poder ser considerado tiempo real.

RNF3 La frecuencia de refresco de la información de los metadatos del vuelo en la interfaz gráfica será de al menos 4Hz.

RNF4 La interfaz de usuario será accesible a desde Linux, Android y Wear OS.

RNF5 Existencia de un dron para la realización de pruebas en condiciones reales.

RNF6 El servidor de retransmisión de vídeo deberá ser capaz de retransmitir a 3 dispositivos conectados simultáneamente.

RNF7 El sistema debe estar acabado y documentado para el 7 de Septiembre.

3.2 Metodología de desarrollo

Esta sección comenzará con una breve introducción a las metodologías ágiles, continuará con la metodología iterativa e incremental

3.2.1 Metodología ágiles

Las metodologías ágiles son aquellas que dan cabida a cambios en los requisitos y cambios en la prioridad de los mismos durante el desarrollo de un producto software.

Los valores en los que se basa el manifiesto ágil son los siguientes [27]:

1. Individuos e interacciones sobre procesos y herramientas
2. Software funcionando sobre documentación extensiva
3. Colaboración con el cliente sobre negociación contractual
4. Respuesta ante el cambio sobre seguir un plan

Además de los valores mencionados anteriormente, este tipo de metodologías siguen un conjunto de doce principios [28] para conseguir el desarrollo ágil:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

3.2.2 Metodología iterativa e incremental

La metodología elegida para el desarrollo de este proyecto fue la metodología ágil 3.2.1 iterativa e incremental. Esta metodología nos permitió realizar un desarrollo continuo, mejorando y añadiendo nuevas funcionalidades sobre la iteración anterior.

En este proyecto el uso de una metodología como la iterativa e incremental permite gestionar los cambios que se van produciendo en las necesidades del proyecto, pudiendo realizar ajustes en los requisitos al principio de cada iteración usando la información de como han transcurrido las iteraciones previas. Además, algunos de los requisitos definidos al comienzo del desarrollo no están del todo claros e indican más una dirección que debe llevar el proyecto, que una característica fija. Otro beneficio del uso de esta metodología es la focalización en el trabajo que más valor aporta al conjunto del proyecto en cada momento.

Para mantener un control de lo que se estaba haciendo en el análisis, planificación y seguimiento del proyecto se han utilizado las aplicaciones Microsoft Planner Apartado 2.2.6 y Draw.io Apartado 2.2.6.

3.2.3 Nuestra implementación de la metodología iterativa e incremental

El desarrollo del proyecto se realizará haciendo iteraciones de una semana de duración, permitiendo de esta forma los cambios rápidos en los requisitos del proyecto.

Tras cada iteración, se hace una reunión con el tutor, en la que se comentan los resultados de la presente iteración y cuales deben ser las tareas a llevar a cabo en las siguientes en base a los resultados de las anteriores. En este punto se podrá acordar tanto la creación nuevos requisitos que vallan surgiendo o la eliminación o reducción de prioridad de algunos de los requisitos ya presentes Apartado 3.1.

3.3 Planificación y seguimiento

Tal y como se menciona en la sección previa, este proyecto será desarrollado usando la metodología iterativa incremental. A continuación se muestran las diferentes iteraciones llevadas a cabo.

3.3.1 Iteraciones

1 Pasos iniciales

En esta iteración mostramos los primeros pasos en el desarrollo de nuestro proyecto. Las tareas llevadas han sido:

T-1.1 Buscar un drone físico para pruebas reales.

T-1.2 Analizar los requisitos iniciales Apartado 3.1 y diseñar el sistema de forma genérica Capítulo 4.

T-1.3 Familiarizarse con la librería Olympe [17].

T-1.4 Instalar el software para desarrollo.

T-1.5 Configurar el simulador Parrot Sphinx [12].

2 Primera versión estación de control

Tras realizar la compra del Parrot Anafi Apartado 2.2.7 y haber diseñado de forma genérica el sistema como se muestra en Apartado 4.1.2, se realizó la segunda iteración con los siguientes objetivos.

T-2.1 Diseñar la estación de control Figura 4.3.

T-2.2 Desarrollar una interfaz básica en pygame[29] que permita el control remoto del drone para la realización de las primeras pruebas.

3 Modelo de datos y streaming

Al finalizar la segunda iteración, se vieron las posibilidades y limitaciones que tenía el SDK de parrot [17], determinando de este modo cuales serían los siguientes pasos.

T-3.1 Modeladar y crear la base de datos y endpoints para guardar información.

T-3.2 Desarrollar el módulo de *live streaming* para video y metadatos del dron (principalmente batería, posición GPS e intensidad de la conexión con el dron).

T-3.3 Estudiar las opciones para el detector del punto de aterrizaje.

4 Primera versión del algoritmo de aterrizaje

En la reunión de la iteración 3 surgió la idea de mejorar la interfaz gráfica para añadir un mapa **RF6**.

T-4.1 Desarrollar el detector del lugar de aterrizaje.

T-4.2 Desarrollar el primer algoritmo de aterrizaje.

5 Interfaz de control y segundo algoritmo de aterrizaje

En la reunión de la iteración 4 se propuso crear una interfaz multidispositivo, haciendo uso de del software de live streaming desarrollado en la iteración 3. También se habló de los fallos vistos en los intentos de aterrizaje del primer algoritmos y se propusieron cambios.

T-5.1 Diseñar la interfaz de control.

T-5.2 Desarrollar interfaz de control para linux en Flutter Apartado 2.2.5.

T-5.3 Desarrollar el segundo algoritmo de aterrizaje.

T-5.4 Desarrollar un sistema sencillo para integrar detectores.

6 Controlador de misiones

En la reunión esta iteración se propuso el algoritmo de aterrizaje 3 como solución a los problemas de los algoritmos 1 y 2.

T-6.1 Desarrollar el controlador de misiones.

T-6.2 Desarrollar de la vista de creación de misiones.

T-6.3 Optimizar el software.

T-6.4 Desarrollar el tercer algoritmo de aterrizaje.

7 Interfaces de usuario en Flutter

T-7.1 Modificar la aplicación de Flutter para que sea responsive.

T-7.2 Portar la aplicación de linux a Android y Wear OS.

T-7.3 Modificar interfaz y controlador de misiones para aceptar varios puntos de aterrizaje y despegue.

8 Memoria del proyecto

El seguimiento de la iteración 8 se dá por finalizado el sistema y se comienza con la documentación del mismo.

T-8.1 Escribir memoria del proyecto.

T-8.2 Crear un canal de *youtube* con videos demostrativos.

3.3.2 Ejecución de las tareas

En la tabla 3.1 se muestra el desglose en horas de las tareas realizadas en cada iteración, así como el reparto según cada rol previsto en el proyecto (ver apartado 3.4.2). En este caso las horas del asesor sólo se reflejan pero no se contabilizan al ser éstos los tutores del proyecto.

3.4 Costes del proyecto

Para el cálculo de los costes totales del proyecto se hizo teniendo en cuenta los costes de material (que podría ser utilizado para otros proyectos) y los costes de los recursos humanos necesarios para su ejecución.

3.4.1 Costes de material

En este apartado se muestran los coste del material necesario para desarrollar este proyecto (tabla 3.2). Se incluye tanto el coste de adquisición como el coste imputable teniendo en cuenta el período de amortización contable y las horas dedicadas al proyecto.

Los elementos críticos han sido el ordenador de desarrollo, que también ha sido usado como PC de control del dron, y el propio dron. Sin embargo, el coste imputable es relativamente bajo al tener periodos de amortización alto. El coste final de material ha sido de 135 euros.

Tabla 3.1: Reparto de las tareas entre los diferentes roles.

Tarea	Horas	Asesor	Jefe	Analista	Programador
Iteración 1	**40**	**3**	**6**	**10**	**24**
T-1.1	4	0	2	0	2
T-1.2	17	0	1	10	6
T-1.3	11	0	1	0	10
T-1.4	3	0	1	0	2
T-1.5	5	0	1	0	4
Iteración 2	**30**	**3**	**5**	**10**	**15**
T-2.1	19	0	4	9	6
T-2.2	11	0	1	1	9
Iteración 3	**45**	**2**	**5**	**10**	**30**
T-3.1	15	0	3	1	11
T-3.2	25	0	1	5	19
T-3.3	5	0	1	4	0
Iteración 4	**30**	**1**	**2**	**4**	**24**
T-4.1	20	0	1	2	17
T-4.2	10	0	1	2	7
Iteración 5	**35**	**1**	**4**	**10**	**21**
T-5.1	5	0	1	4	0
T-5.2	15	0	1	2	12
T-5.3	5	0	1	2	2
T-5.4	10	0	1	2	7
Iteración 6	**50**	**1**	**4**	**8**	**38**
T-6.1	18	0	1	2	15
T-6.2	18	0	1	2	15
T-6.3	8	0	1	2	5
T-6.4	6	0	1	2	3
Iteración 7	**30**	**1**	**3**	**7**	**20**
T-7.1	12	0	1	2	9
T-7.2	7	0	1	2	4
T-7.3	11	0	1	3	7
Iteración 8	**88**	**11**	**22**	**44**	**22**
T-8.1	80	10	20	40	20
T-8.2	80	1	2	4	2
Total	**348**	**23**	**51**	**103**	**194**

Tabla 3.2: Coste del material.

Material	**Coste adquisición (€)**	**Coste imputable (€)**
Ordenador	950	50
Parrot Anafi	500	50
Plataforma Aruco	10	10
Antena Wifi	25	25
Total	1.485	135

Tabla 3.3: Coste del personal.

Rol	**Tarifa (€/hora)**	**Tiempo (h)**	**Coste (€)**
Jefe de Proyecto	70	51	3.570
Analista	40	103	4.120
Programador	25	194	4.850
Total	–	348	12.540

3.4.2 Coste del personal

Los costes del personal del proyecto se pueden ver en la tabla 3.3. Se han tenido en cuenta el desglose por roles de las distintas tareas realizadas (ver apartado 3.3.1). Como ya se ha comentado anteriormente, no se computan las horas de los asesores, papel que recae en los tutores del trabajo fin de grado.

Como era de esperar, el coste del personal es la partida más importante, alcanzando un total de 12.450 euros.

Capítulo 4

Diseño

En este capítulo se verá en primer lugar la arquitectura propuesta para cumplir con los requisitos Apartado 3.1. A continuación se detallarán la interfaz propuesta para el control del sistema y el modelo de datos usado para guardar información de las misiones y vuelos. Finalmente, se comentarán los patrones de diseño más relevantes usados en el desarrollo.

4.1 Arquitectura del sistema

Para explicar la arquitectura del sistema vamos a hacer uso del modelo C4. El modelo C4 consiste en un conjunto jerárquico de diagramas de arquitectura de software para contexto, contenedores, componentes y código, que permiten explicar la arquitectura del sistema con diferentes niveles de detalle a personas con diferentes niveles de conocimiento.

4.1.1 Diagrama de contexto

El diagrama de contexto proporciona una visión general de como interactúa el sistema que desarrollamos con las personas y otros sistemas ya existentes. En nuestro caso, tal como se puede ver en la figura 4.1 el dron será controlado por el sistema de control, que a su vez será gestionado por un humano.

4.1.2 Diagrama de contenedores

El diagrama de contenedores profundiza en el nivel de detalle del sistema que vamos a crear, permitiendo visualizar las diferentes bases de datos, aplicaciones y servicios que compondrán el sistema final.

- **Interfaces gráficas**: para el desarrollo de las interfaces gráficas de linux, Android y Wear OS se decidió usar Flutter 2.2.5. Flutter nos brinda la posibilidad de crear aplicaciones para diferentes plataformas con un mismo código, o con modificaciones muy

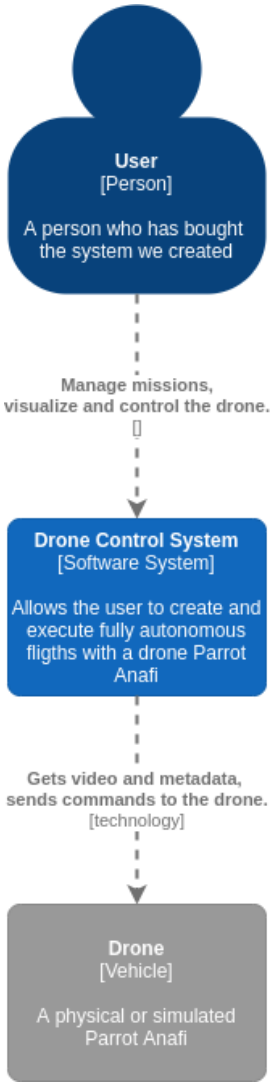


Figura 4.1: Diagrama de contexto de nuestro sistema.

pequeñas del mismo. En nuestro caso, Linux y Android compartirán casi todo el código, y la versión para Wear OS será una versión limitada de las anteriores.

- **API HTTP**: se encarga de crear y recuperar información de misiones y drones.
- **Broker MQTT**: el protocolo MQTT proporciona un menor consumo de ancho de banda, menor latencia que el de Hypertext Transfer Protocol. (HTTP) y además, puesto que se va a realizar comunicación con varios dispositivos, es una mejor solución que usar un protocolo como Websockets. El *broker*, que en nuestro caso será Mosquitto 2.2.6, es el módulo encargado de realizar la redistribución de los metadatos del dron.
- **Control station**: es el contenedor más importante de nuestro sistema y está desarro-

llado en Python [13], por estar escrita en ese lenguaje la librería de control del dron olympe [17] de Parrot y por ser un buen lenguaje para desarrollar un sistema de estas características en un tiempo limitado. Nos sirve de puerta al dron, siendo sus principales tareas la recepción de información directamente del dron, reenvío de la misma al resto de módulos, y control del propio dron. Dentro de este contenedor se encuentran implementados los algoritmos de detección del punto de aterrizaje, los de aterrizaje y los de control de misión.

- **Database**: el sistema desarrollado, utilizará PostgreSQL con la extensión Postgis [25] para poder almacenar datos espaciales, tal y como se mencionó en el apartado Tecnologías utilizadas.

- **Detector**: se conecta con el servidor de vídeo para procesarlo y devuelve el vídeo postprocesado. El realizar la integración de este modo permite tener el detector en máquinas distintas al servidor de vídeo y a la estación de control.

- **Media Server**: se encarga de reenviar el vídeo recibido del dron a los diferentes dispositivos que estén conectados. Puede estar instalado en una máquina diferente a la estación de control, de modo reduzca la carga de procesamiento en ésta.

4.1.3 Diagrama de componentes

Tal y como se mencionó en el apartado anterior, el sistema de control es el contenedor encargado de comandar el dron y está compuesto por los siguientes componentes (figura 4.3):

- **Drone**: mantiene la conexión con el dron físico. En caso de pérdida de la conexión, la reconexión al dron se realizará desde este mismo componente.

- **Landing Detector**: se encargará de usar las imágenes recibidas del dron en el momento de aterrizaje para procesarlas en búsqueda del punto de aterrizaje.

- **Landing Controller**: haciendo uso del punto detectado por el *landing detector*, el *landing controller* determinará qué movimientos debe realizar el dron para converger en el punto de aterrizaje. El movimiento será indicado al dron físico a través de nuestro componente Drone.

- **Remote Controller**: es el componente encargado de exponer el control del sistema a las interfaces gráficas mediante el uso de un *websocket*. Un *websocket* nos proporciona comunicación bidireccional, permitiendo tanto la recepción de órdenes desde la interfaz como el envío de información a la misma.

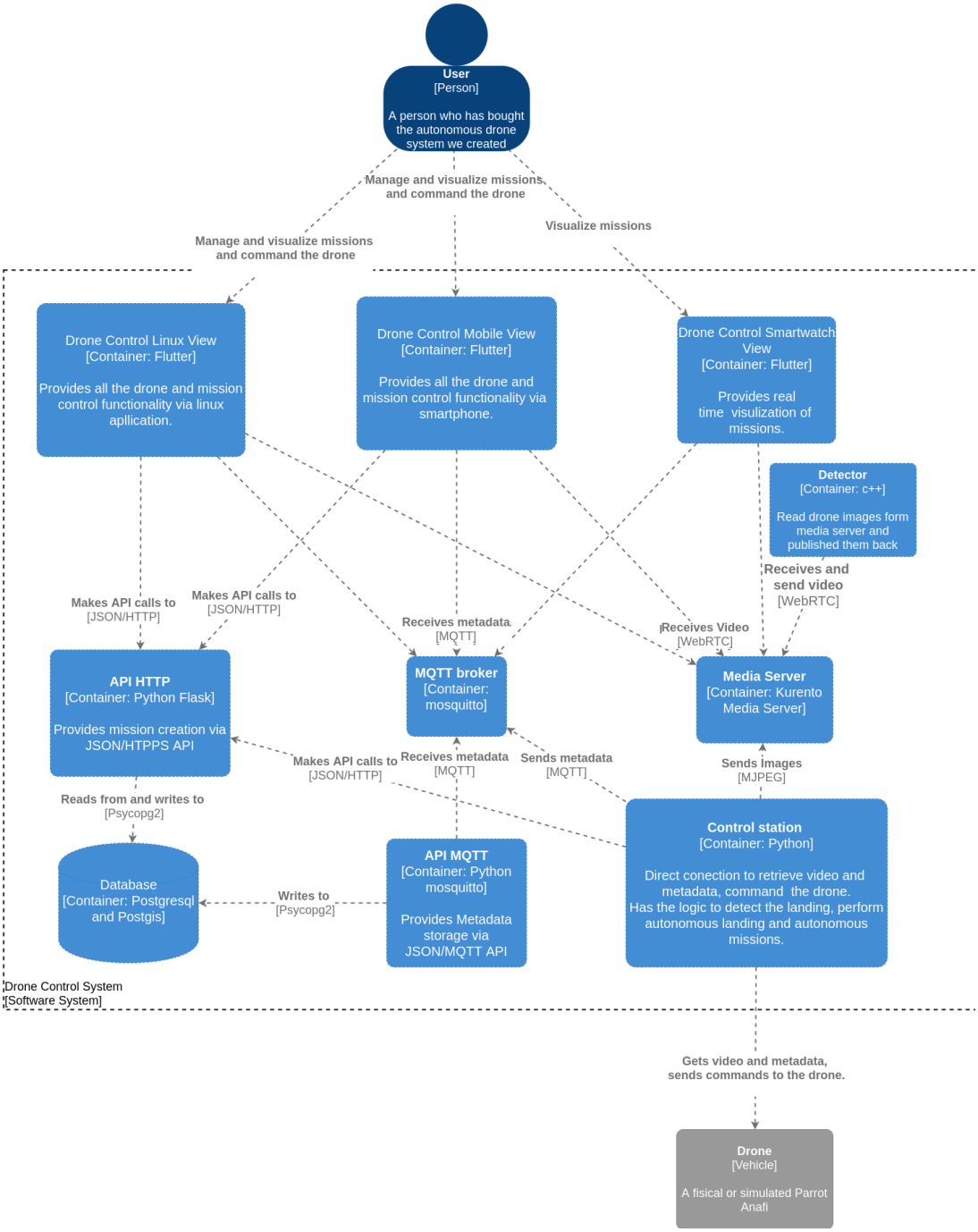


Figura 4.2: Diagrama de contenedores de nuestra propuesta.

- **Live Module**: comparte la imagen en tiempo real con el servidor de vídeo y publica los metadatos recibidos del dron en el Broker MQTT.
- **Mission Controller**: controla al dron durante la ejecución de una misión.

Cada componente implementa una interfaz que es usada por los demás componentes para la comunicación. Esta distribución de componentes incrementa la modularidad de la solución y permite el cambio en un componente sin afectar a los demás.

4.2 Interfaz de usuario

La interfaz de usuario es una de las partes más importantes de un producto software, que marca la diferencia entre el éxito o el fracaso del mismo. Es por ese motivo, que en el presente proyecto se han desarrollado tres interfaces que permiten al usuario final crear, controlar y visualizar un dron en misiones de vigilancia o reconocimiento.

Como se pretendía mantener un mismo diseño estético que fuera accesible desde sistemas operativos basados en Linux, Android y Wear OS (ampliable en un futuro) se decidió tomar la decisión de usar Flutter, que tal y como se menciona en el apartado 2.2.5 es compatible con esas plataformas (y otras muchas).

4.2.1 Interfaz Linux

La interfaz de Linux está pensada para permitir un control total del sistema. A continuación se describen las principales vistas:

- **Pantalla principal**: permite navegar a la pantalla de visualización de misiones o la pantalla de control del dron.
- **Pantalla de visualización de misiones**: muestra en una tabla las misiones que hay actualmente creadas y un botón para acceder al planificador.
- **Pantalla de creación misiones**: permite crear las misiones en un mapa, pudiendo añadir un nombre y una descripción a la misión, además de los puntos por los que debe pasar el dron y si son de aterrizaje o despegue.
- **Pantalla de control del dron**: desde esta pantalla se puede visualizar en tiempo real la imagen del dron y la imagen procesada. También se visualiza la información que se considera más relevante para la misitón: batería restante, calidad de la conexión, altura actual del dron y un mapa en donde se muestra la posición GPS del mismo. Usando el teclado se puede aterrizar, despegar, tomar el control del movimiento del dron, cancelar misiones, girar la cámara, etc.

En la figura 4.4 se puede ver un diagrama de las pantallas accesibles desde la interfaz Linux.

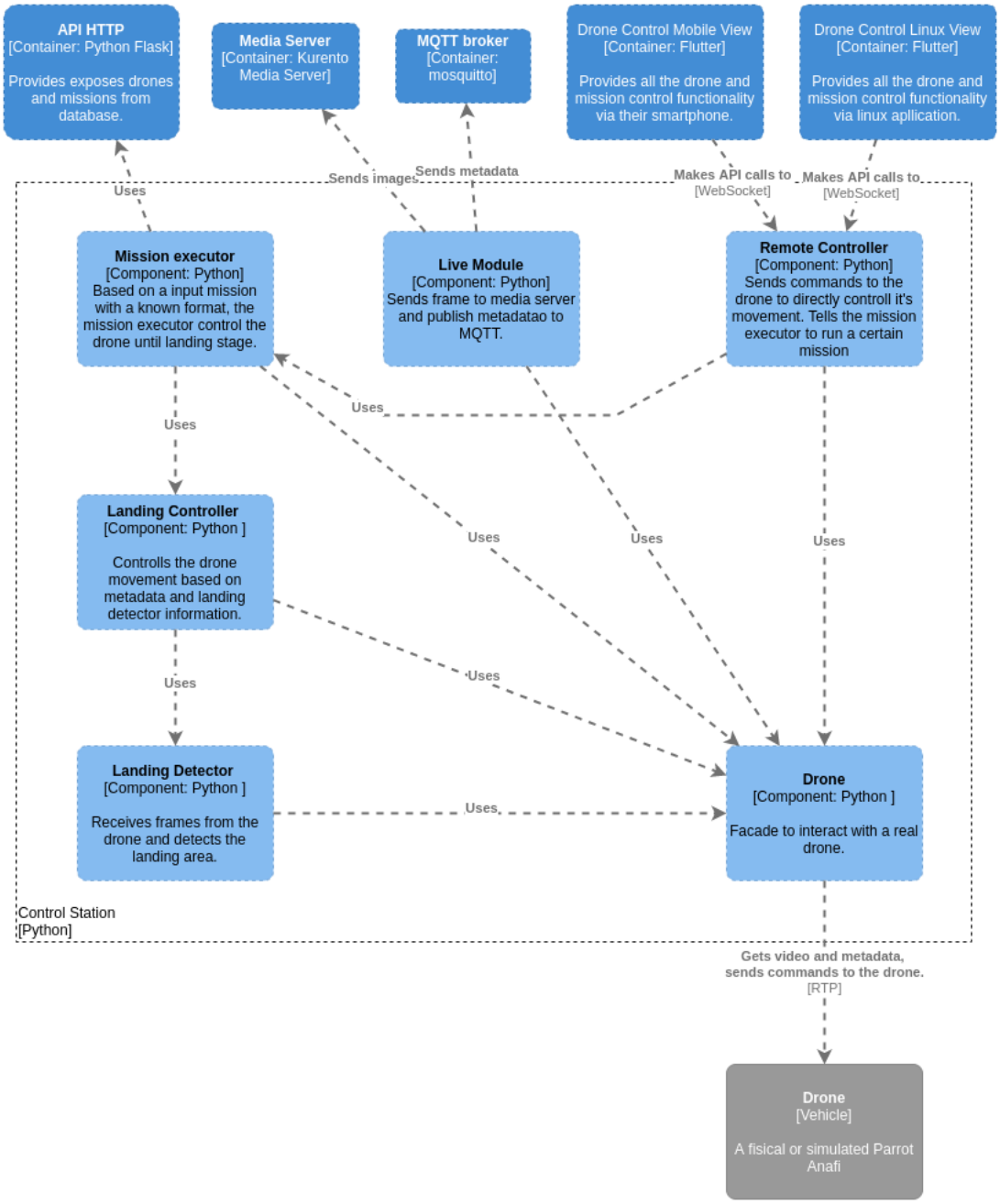


Figura 4.3: Diagrama de componentes.

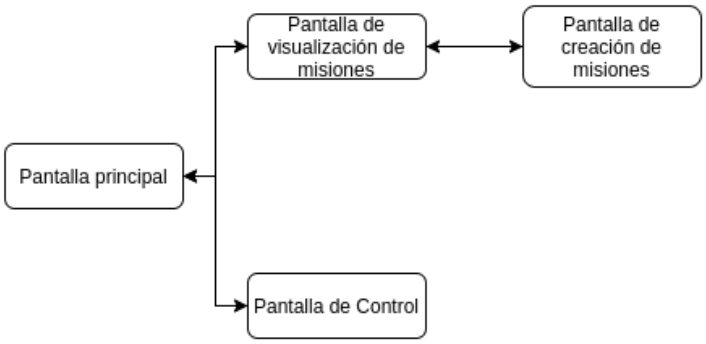


Figura 4.4: Diagrama de pantallas para los sistemas operativos Linux y Android.

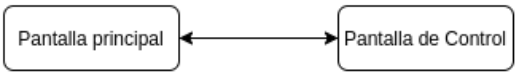


Figura 4.5: Diagrama de pantallas para la versión Wear OS.

4.2.2 Interfaz Android

Se plantea como una interfaz desde la que crear una misión rápidamente y visualizarla. Tiene las mismas opciones que la aplicación para linux, aunque no se permite el control de el dron por teclado. El diagrama puede ser consultado en 4.4.

4.2.3 Wear OS

La creciente popularidad de los Smartwatch, ha hecho que el desarrollo de una aplicación para este tipo de dispositivos pueda marcar una diferencia entre este producto y los vistos en el apartado 2.1.

La versión de Smartwatch es, por motivos de tamaño, una versión reducida de las interfaces de Linux y Android. Ésta se limita a la visualización de misiones en tiempo real, pudiendo ver la imagen del dron, la imagen una vez procesada o el mapa. En la misma vista que estas tres opciones se encuentra, una vez más, información sobre la batería, la calidad de la conexión y la altura de vuelo. En la figura 4.5 se puede ver un diagrama de pantallas en el Smartwatch.

4.3 Modelo conceptual de datos

Como ya se ha mencionado en el Capítulo 2, en este proyecto se hace uso del sistema de gestión de bases de datos de código libre PostgreSQL [24] junto con Postgis [25] y está desplegada en el interior de un *docker*, lo cual facilita replicar o migrar la base de datos a otra máquina en caso de ser necesario.

El esquema utilizado para la persistencia de datos se puede consultar en la figura 4.6. Se han identificado cinco entidades:

- **dron**: dispone de un modelo, una descripción y un tiempo máximo de vuelo. Este último atributo permitirá en un futuro calcular si un vuelo se puede o no realizar.
- **mission**: esta entidad almacena el nombre y la descripción de las misiones que puede ejecutar el dron.
- **mission_stages**: en ella se almacenan las fases de las que está compuesta cada misión. Tiene los campos "command" para especificar que debe de hacer el dron y "route", para especificar una geometría a usar con el comando. Los comandos aceptados por el sistema de ejecución de misiones (sección 5.1.4) son:
  - **TAKE_OFF**: despega el dron y lo mueve a la posición de "route".
  - **GO**: mueve al dron a la posición de "route".
  - **LAND|<tagId>**: mueve al dron a la posición de "route" y lo aterriza en el marcador especificado.
- **flights**: relaciona información de los vuelos con drones y misiones.
- **flight_metadata**: contiene información de los vuelos. Desde ninguna de las APIs mencionadas en la sección 4.1.2 se puede acceder a esta información. Sirve para poder ser consultada desde un sistema geográfico como QGUIS, en caso de tener un problema o perder el dron. Sirvió durante el desarrollo para revisar la información de los vuelos, y podría servir para determinar donde se ha perdido el contacto con el dron en caso de accidente.

4.4 Patrones importantes

En el desarrollo de este proyecto se han usado varios patrones de diseño. De entre todos ellos, destacan en patrón Micro Air Vehicle Link (BLOC), usado en le *frontend*, y el patrón observador en la estación de control.

4.4.1 Patrón BLOC

El patrón BLOC permite mantener el estado de una aplicación Flutter. Consta de tres componentes:

- **Bloc**: es el cuerpo del patrón y es el encargado de reaccionar a los eventos devolviendo un nuevo estado en caso de que éste haya cambiado. Funciona de forma asíncrona.

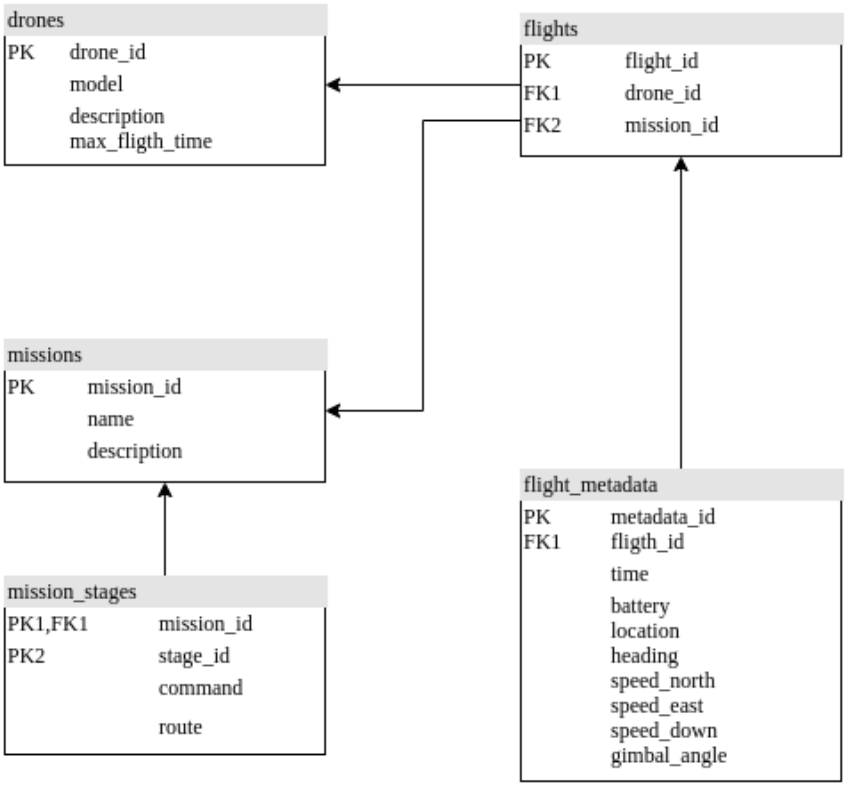


Figura 4.6: Esquema de la base de datos de nuestro sistema.

- **Events**: se suelen lanzar desde la interfaz cuando el usuario hace click en un botón, hace *scroll* y llega a un determinado punto desde el que hay que pedir más información, etc. Ejemplos de eventos pueden ser: Connect(ip), Disconect(), FetchMissions(), etc.

- **State**: es el cambio que devuelve el patrón BLOC y puede contener información en su interior para ser mostrada en la interfaz de usuario. Ejemplos de estado son: Connected, Disconected, Error, etc.

En la figura 4.7 se puede ver un diagrama sobre el funcionamiento de este patrón.

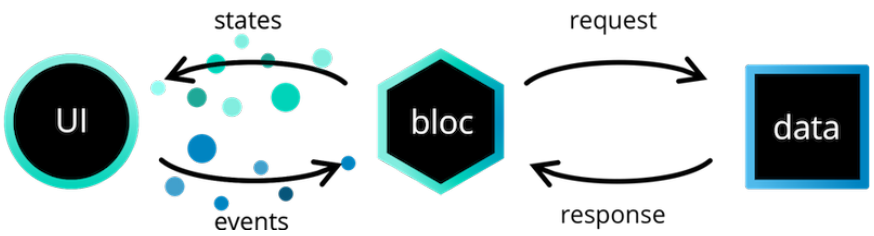


Figura 4.7: Ejemplo de patrón BLOC.

4.4.2 Patrón observador

El patrón observador permite a un objeto notificar a otros sobre cambios en su estado. En nuestro caso particular, la imagen del dron y los metadatos serán utilizados por diferentes módulos como el de aterrizaje y el de *streaming*. Y en un futuro la imagen podría ser usada por otro módulo que detecte objetos y evite accidentes.

Capítulo 5

Implementación y pruebas

Este capítulo comenzará explicando algunos de los problemas y peculiaridades que surgieron durante la implementación de la arquitectura y terminará describiendo las pruebas realizadas con el sistema y qué aprendimos de ellas.

5.1 Implementación

5.1.1 Detector del punto de aterrizaje

Aunque finalmente usamos la librería Aruco [16], hubo un proceso de experimentación previo en el que se probaron varios detectores de creación propia.

Inicialmente se pensó en hacer un detector basado en el espacio de colores Hue Saturation Value. (HSV), filtrando en un primer paso por colores deseados en un determinado rango, y en un segundo por las formas de los contornos resultantes del paso previo.

Como el resultado conseguido variaba dependiendo del tono de la luz incidente sobre la plataforma, reflejos, etc, se trató de mitigar este problema realizando modificaciones dinámicas a los ajustes de exposición y balance de blancos de la cámara a través del driver "v4l2" de Linux. Aunque el resultado tras esta modificación era mejor y se admitía una detección parcial de cerca del 60% en condiciones buenas de luz, seguía habiendo casos en las que la incidencia de la luz creaba reflejos o modificaciones en el tono de los colores que impedían la detección de la plataforma.

La librería Aruco [16] permite la detección de marcadores como los mostrados en la figura 2.5. Los marcadores se pueden crear con formas personalizadas, especificando su forma mediante una matriz de ceros y unos, o se pueden usar diccionarios ya definidos por esta librería. Esta matriz está siempre rodeada de un borde negro, lo que hace que la implementación de los algoritmos de detección del marcador sea muy eficiente.

En nuestro caso usamos el marcador número dos del diccionario de 5x5 que se puede ver en figura 5.1. Nótese que para facilitar el transporte del marcador, éste está forrado sobre una

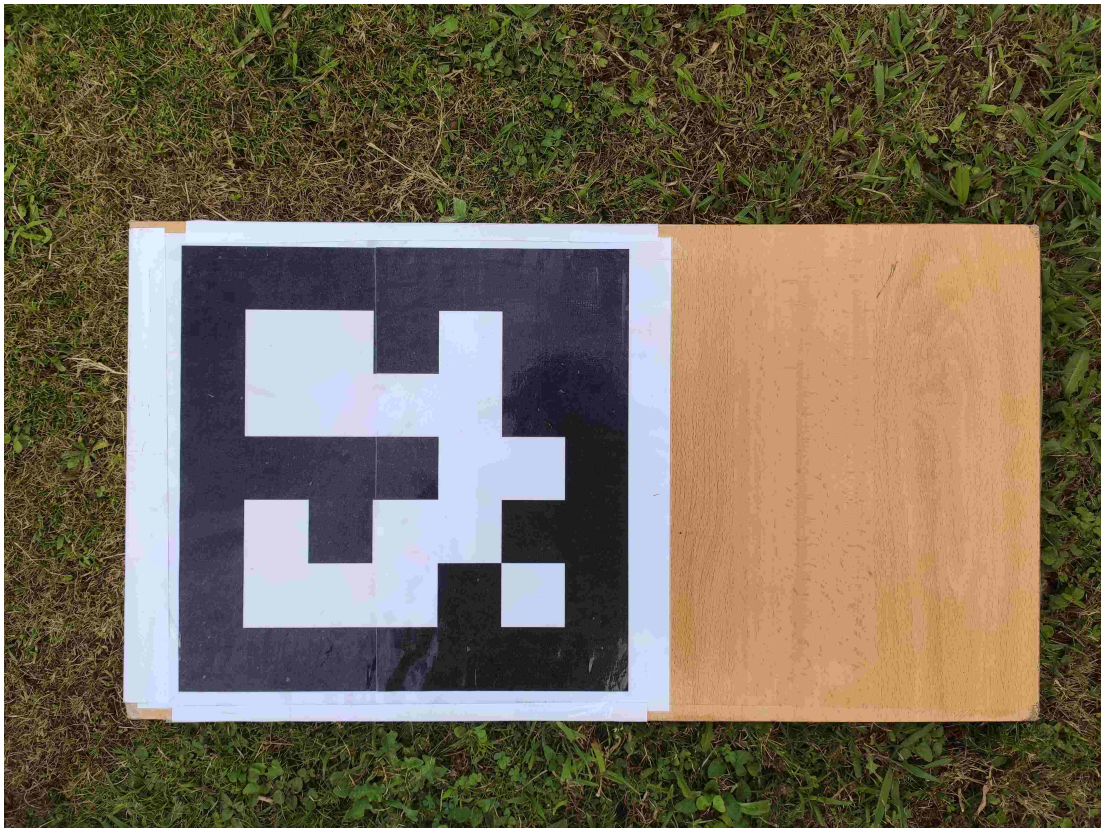


Figura 5.1: Marcador número dos del diccionario de 5x5 usado en este proyecto.

tabla. Lo ideal sería poder pintar el marcador con colores mate, que no causen reflejos como los vistos en la imagen.

Este tipo de marcadores nos permite calcular la rotación y posición de la cámara respecto del marcador en el espacio, lo cual es una ventaja si lo comparamos con los detectores de creación propia que habíamos probado previamente. La detección de los marcadores Aruco se muestran en el algoritmo 5.1.

5.1.2 Servidor de retransmisión de vídeo

Para no complicar el desarrollo, en lugar de configurar un servidor externo al sistema de control para retransmitir el vídeo del dron a las diferentes interfaces que lo pueden consumir, se hizo una clase en python (ver código 5.2) que permite retransmitir imágenes de opencv en formato Motion JPEG. (MJPEG). Para poder integrarlo de forma sencilla, dicha clase se ejecuta en un hilo distinto al principal, de modo que no se interrumpe la ejecución mientras que se aceptan y sirven nuevas peticiones del vídeo.

Cabe recalcar que este apartado difiere de lo indicado en el Capítulo 4, y si bien no afectó al desarrollo del proyecto, consideramos relevante modificar la implementación de este servidor

```
parameters = aruco.DetectorParameters_create()
dictionary = aruco.Dictionary_get(aruco.DICT_5X5_1000)
...
#get frame from image or video
frame=cv2.imread
...
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray,
    dictionary, parameters=parameters)
#use corners and ids
...
#draw detected markers in image
frame = aruco.drawDetectedMarkers(frame, corners, borderColor=(0,
    255,0))
...
```

Algoritmo 5.1: Código python para la detección de Aruco markers.

para cumplir con lo especificado en el diagrama de contenedores Figura 4.2 antes de usar el sistema en un entorno de producción. Teniendo aislado en una máquina física solo al sistema de control, podríamos aumentar el número de consumidores de la imagen del dron sin dañar el rendimiento del sistema.

5.1.3 Algoritmo de aterrizaje

El desarrollo del algoritmo de aterrizaje fue un proceso iterativo de prueba y error. A continuación de describe como se hizo el desarrollo para llegar a la solución actual.

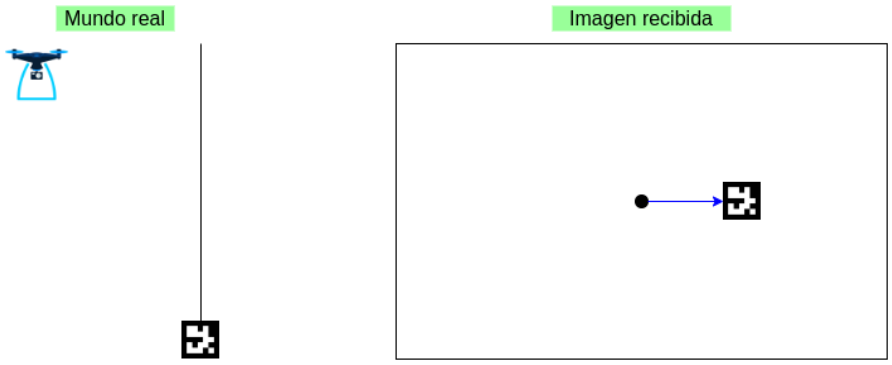
Los drones suelen tener una limitación en la altura de vuelo que hace que no se pueda bajar de dicha altura sin activar el modo de aterrizaje. Esto es lo que ocurre en el caso del Parrot Anafi (Apartado 2.2.7). Por este motivo, los algoritmos descritos a continuación terminan todos activando el modo *landing* del dron en caso de estar centrado, siempre y cuando su altura sea menor a 90 cm.

1. **Algoritmo 1**: Se plantea como un desplazamiento continuo en un plano XY paralelo al suelo en dirección al punto de aterrizaje haciendo giros de *pitch* y *roll* en el dron (ver sección 2.3.1). Y finalmente un movimiento vertical cuando estamos cerca de la perpendicular del punto de aterrizaje.

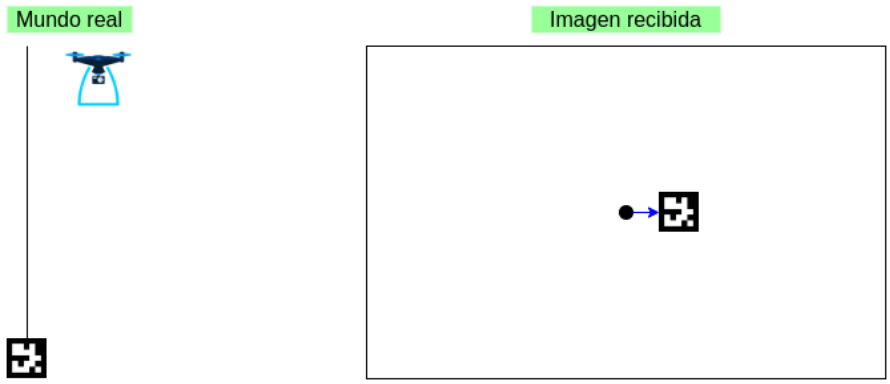
2. **Algoritmo 2**: Es idéntico al 1 excepto que el movimiento en el plano XY se realiza usando *pitch* y *yaw*. Se trata de "alinear" el dron con el punto de aterrizaje girándolo sobre el eje Z aplicando una rotación en *yaw*. Es decir, la idea es poner (casi) todo el error en el eje X y después reducirlo aplicando *pitch*. De nuevo, al estar cerca de la perpendicular del punto de aterrizaje se inicia el descenso sobre el eje Z.

```
class Endpoint(object):

    def __init__(self,fps=30,image_shape=(1280, 720)):
        self.fps = fps
        self.image_shape = image_shape
        image = np.zeros((image_shape[0],image_shape[1],3),np.uint8)
        self.set_image(image)

    def set_image(self, image):
        ret, jpeg = cv2.imencode('.jpg', cv2.resize(image,
                                                    self.image_shape))
        frame = jpeg.tobytes()
        self.image = (b'--frame\r\n'
            b'Content-Type:image/jpeg\r\n'
            b'Content-Length: ' + f"{len(frame)}".encode() + b'\r\n'
            b'\r\n' + frame + b'\r\n')

    def generator(self):
        while True:
            sleep(1/self.fps)
            yield self.image

    def __call__(self, *args):
        return Response(self.generator(), mimetype=
                        'multipart/x-mixed-replace; boundary=frame')


class LiveStreamingServer(Thread, object):

    def __init__(self, host="0.0.0.0", port=5001):
        self.port = port
        self.host = host
        self.endpoint = Endpoint(fps=30,image_shape=(640,360))
        self.app = Flask("dronLiveStreaming")
        self.app.add_url_rule("/live_streaming", "live_streaming",
                              self.endpoint)

        #Start thread
        super().__init__()
        super().start()

    def set_image(self, image):
        self.endpoint.set_image(image)

    def run(self):
        self.app.run(host=self.host, port=self.port, debug=False)
```

Algoritmo 5.2: Clase Python para retransmitir el vídeo del dron a las diferentes interfaces.



(a) Antes de cruzar la vertical con el punto de aterrizaje.



(b) Después de cruzar la vertical con el punto de aterrizaje.

Figura 5.2: Imagen demostrativa de la latencia del dron durante el proceso de aterrizaje.

3. **Algoritmo 3**: Modificación del algoritmo 1 pero interrumpiendo el movimiento en el plano XY cuando comienza el descenso.

De los algoritmos descritos anteriormente, el 1 y el 2 presentan un problema que hace que el dron se empiece a balancear de un lado al otro de la perpendicular de la plataforma de aterrizaje. Tal y como se muestra en la tabla Cuadro 5.1, el tiempo entre que una imagen es tomada y llega al dron supera los 200 milisegundos. Este retraso en la recepción de la imagen produce que se le envíen movimientos con el mismo retraso cuando físicamente acaba de cruzar la vertical con el punto de aterrizaje.

El problema puede ser visualizado gráficamente en la Figura 5.2. El algoritmo 3 ataja dicho problema de un modo sencillo y que en la práctica demostró ser el más efectivo de todos los probados. Los resultados obtenidos en el algoritmo 3 se pueden ver en el apartado 5.2.3.

5.1.4 Sistema que ejecuta misiones

Para ejecutar las misiones guardadas en la base de datos, se hizo uso de una clase 4.1.3 desde la que se puede indicar el número de la misión (consultable desde las interfaces) en el método "set_mission", desde este punto se recuperarán las etapas que componen la misión seleccionada y se iniciará la función "self.run" en un thread. Ejecutar la misión en un hilo permite que el programa principal no quede bloqueado, de modo que la misión podría ser cancelada en cualquier momento por el usuario.

En este método, se van quitando de una lista las distintas etapas de la misión y una a una se van ejecutando y esperando a que acaben. Como ya se mencionó en el modelo de datos 4.3, en cada etapa se indica un comando y una posición. Estos dos valores serán usados para determinar qué debe hacer el dron en cada momento.

Actualmente se contemplan tres opciones:

1. **TAKE_OFF**: realiza el despegue del drone luego tras realizar una espera de 5 segundos y inicia el desplazamiento hacia la posición indicada.

2. **GO**: indica al dron la posición GPS a la que debe de moverse.

3. **LAND|id**: desplaza al dron a la posición indicada en "route", espera a que se llegue a ese punto y después inicia el descenso hasta una altura de 6 metros. Una vez llegado a este punto, se cede el control de la misión al *landing controller*, que será el encargado de realizar la fase final del aterrizaje.

5.1.5 Detectores

Los detectores toman como entrada las imágenes publicadas en el servidor de vídeo (apartado 5.1.2) para procesarlas y publicarlas en un servidor de MJPEG escrito en C++. En el algoritmo 5.4 se muestra la clase principal del detector. Como se puede ver, basta con hacer cualquier modificación en la imagen y publicarla, para que ésta se pueda ver en la interfaz de usuario.

5.1.6 Programación de misiones

Para realizar la programación de misiones se hizo uso de *cron*, el programador de procesos de Linux. El ejecutable de la estación de control admite como parámetro el identificador de la misión (*id*) a ejecutar. En el código mostrado en el algoritmo 5.5 se puede ver como se programa una misión para ser ejecutada cada 2 minutos. Una demostración de esta funcionalidad puede ser consultada en vídeo en [30].

```
class MissionExecutor():

    ...

    def set_mission(self,mission_id):
        self.stages=getStagesFromMission(mission_id)
        self.keep_running=True
        Thread(target=self.run,name="MissionExecutor").start()


    def run(self):
        while len(self.stages)>0 and self.keep_running:
            self.__wait_movement_finished()
            stage=self.stages.pop(0)
            if stage.command.startswith("TAKE_OFF"):
                print("STARTING TAKE_OFF")
                # It seems there is a bug on parrot olympe that
                #    doesn't let you take_off if you jut landed,
                sleep(5)    #  so we wait a bit
                self.drone.drone(TakeOff(_no_expect=True))
                self.drone.moveTo(stage.route[0],
                                  stage.route[1], stage.route[2])

            elif stage.command.startswith("GO"):
                print(f"GO   {stage.route}")
                self.drone.moveTo(stage.route[0],
                                  stage.route[1], stage.route[2])

            elif stage.command.startswith("LAND"):
                print(f"GO LANDING POSITION   {stage.route}")
                self.drone.set_gimbal(-90)
                self.drone.moveTo(stage.route[0],
                                  stage.route[1], stage.route[2])
                self.__wait_movement_finished()

                print(f"DESCENDING TO PRECISSION_LANDING
                        POSITION   {stage.route}")
                self.drone.moveTo(stage.route[0], stage.route[1], 6)
                self.__wait_movement_finished()

                self.drone.movement_finished=False
                self.landing_controller.set_tagId(
                                  int(stage.command.split("|")[1]))
                self.drone.set_landing_controller(
                                  self.landing_controller)
                self.drone.drone.start_piloting()
                self.__wait_movement_finished()

        self.end_mission()
        if self.is_scheduled_mission:
            os._exit(0)
```

Algoritmo 5.3: Clase encargada de la ejecución de las misiones en nuestro sistema.

```
#include <opencv4/opencv2/highgui.hpp>
#include "MJPEGWriter.h"
#include "DetectorYolo.h"

int main(){
  DetectorYolo detector=DetectorYolo("../config/yolo.cfg",
                                     "../config/yolo.weights",
                                     "../config/obj.names");
  MJPEGWriter mjpegWriter(7777);
  VideoCapture cap("http://localhost:11111/live_streaming");     ///
    Points to drone live_streaming
    Mat frame;
  cap >> frame;
  mjpegWriter.write(frame);
  frame.release();
  mjpegWriter.start();
  while(cap.isOpened()){
    cap >> frame;
    // PROCESS FRAME
    detector.detect(frame);

    // NON INTENSIVE "PROCESSOR" FOR TESTING WITHOUT BURNING
    CONTROL STATION
    //  cv::cvtColor(frame,frame,cv::COLOR_BGR2RGB);

    mjpegWriter.write(frame);
    frame.release();
  }
  mjpegWriter.stop();
  exit(0);
}
```

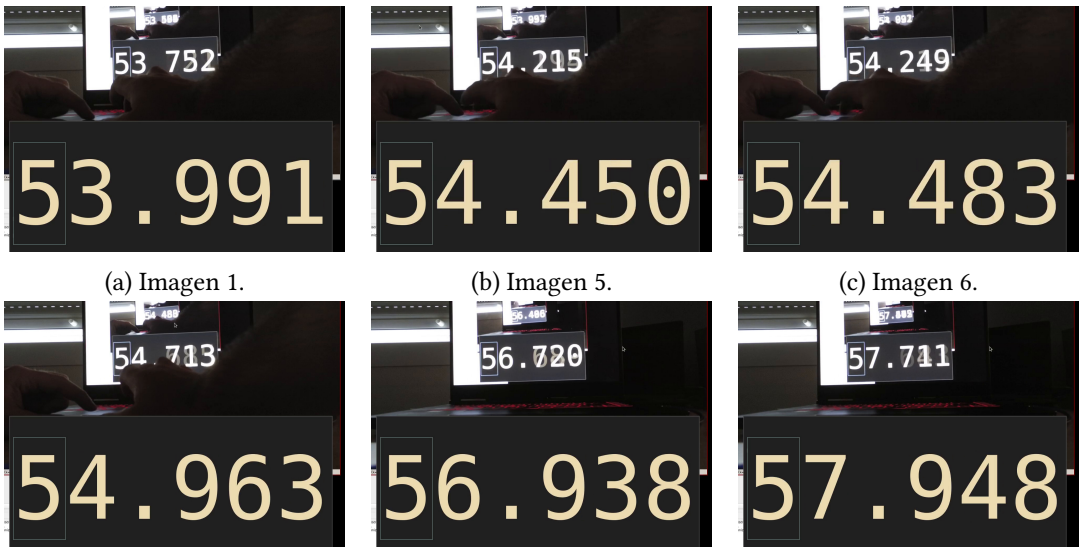
Algoritmo 5.4: Clase encargada de la ejecución del detector

```
*/2 * * * * zsh -l -c '. /path_to_olympe_environment && .
    /path_to_tfg_virtualenv && python
    /path_to_control_station_main.py  missionId ' >>
    /path_to_redirected_output_file 2>&1
```

Algoritmo 5.5: Ejemplo de programación de una misión del dron usando el programador de procesos *cron* de Linux

```
$ while true; do echo -ne "`date +"%S.%3N"`\r"; done
```

Algoritmo 5.6: Comando usado para visualizar un cronómetro de milisegundos en la pantalla del ordenador de control.



(a) Imagen 1. (b) Imagen 5. (c) Imagen 6.

(d) Imagen 9. (e) Imagen 18, mínima latencia. (f) Imagen 20, máxima latencia.

Figura 5.3: Varios ejemplos de imágenes usadas para el cálculo de la latencia.

5.2 Pruebas

En este apartado se describen las principales pruebas realizadas: la latencia de la retransmisión del vídeo desde el dron, el tiempo de procesamiento del vídeo, el aterrizaje automático y varios puntos de aterrizaje.

5.2.1 Latencia retransmisión del vídeo

Objetivo

Determinar el tiempo que transcurre entre que una imagen (cada imagen de vídeo) se captura en el dron y llega al ordenador de control.

Metodología

Se muestra en la pantalla del ordenador de control un cronómetro con precisión hasta los milisegundos (ver algoritmo 5.6) y se apunta con la cámara del dron hacia la pantalla.

Una vez alcanzada esa disposición de los componentes se graba la pantalla durante varios segundos. En cada imagen aparece tanto el cronómetro como la imagen capturada por

Tabla 5.1: Tiempos de latencia de la retransmisión del vídeo desde el dron hasta el ordenador de control de misión.

Imagen	Real (ms)	Captura (ms)	Diferencia (ms)
1	991	752	239
2	431	192	239
3	785	554	231
4	986	752	234
5	450	215	235
6	483	249	234
7	685	450	235
8	713	483	230
9	963	713	**250**
10	523	282	241
11	822	584	238
12	853	612	241
13	261	023	238
14	322	084	238
15	343	115	228
16	522	290	232
17	924	689	235
18	938	720	**218**
19	975	749	226
20	948	711	237
Media	–	–	235.4 ± 6.8

el dron y representada en la pantalla (figura 5.3). En esta prueba se ha usado la aplicación *SimpleScreenRecorder*.

A partir del vídeo, podemos extraer frames y apuntar las diferencias entre el cronómetro en tiempo real en aquellos en los que el cronómetro no se ve borroso. El vídeo puede ser consultado en [31].

Resultados

Los resultados obtenidos en las diferentes pruebas se pueden observar en la tabla 5.1. Se han elegido ciertas imágenes representativas de cada vídeo y se ha apuntado tanto el tiempo real mostrado por el cronómetro como el tiempo de la imagen de la captura del dron. Después se ha calculado la diferencia para cada imagen.

De acuerdo estos datos el tiempo medio transcurrido es de 234.95 ms con una desviación típica de 6.61 ms, con un valor máximo de 250 ms y un mínimo de 218 ms (un rango de 32 ms, es decir, menos de un 15%). La latencia es bastante estable (con un error de menos de un 3% sobre el valor medio), aunque su valor medio es superior a lo deseable.

A pesar de todo ello, la latencia en la retransmisión del vídeo del dron no ha afectado excesivamente a los resultados del proyecto: hemos podido realizar aterrizajes autónomos y detección de objetos procesando remotamente (en el ordenador de control) el vídeo del dron.

Tabla 5.2: Tiempos del procesamiento con un detector tipo YOLO v3 de las imágenes retransmitidas por el dron.

Imagen	Tiempo de procesado (ms)
1	**26**
2	**28**
3	28
4	27
5	26
6	26
7	27
8	28
9	28
10	27
11	27
12	27
13	26
14	28
15	26
16	27
17	27
18	28
19	28
20	28
Media	27.15 ± 0.18

5.2.2 Tiempo de procesamiento de imágenes

Objetivo

Determinar el tiempo transcurrido entre que se recibe una imagen y que se procesa con un detector bastante potente (y pesado) tipo YOLO v3, en lo que podría ser un ejemplo de misión real de vigilancia o reconocimiento.

Metodología

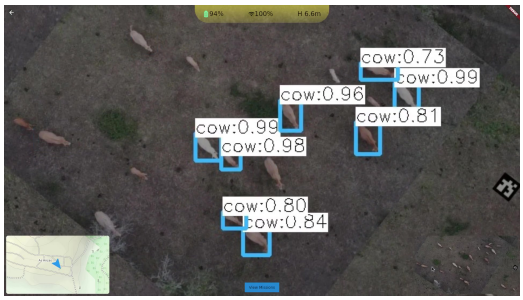
Se captura el tiempo en milisegundos desde el inicio del tiempo en linux (01/01/1970) antes y después de usar el detector y se apunta la diferencia. Se midió el tiempo de procesado de un detector YOLO V3 entrenado con el dataset COCO[32] y con una versión entrenada por nosotros con imágenes del dron en la cual se detectan vacas.

El entrenamiento y procesamiento se ha realizado en el ordenador de control, un portátil con un procesador i7-9750H, 16GB de memoria RAM y una tarjeta gráfica RTX2060.

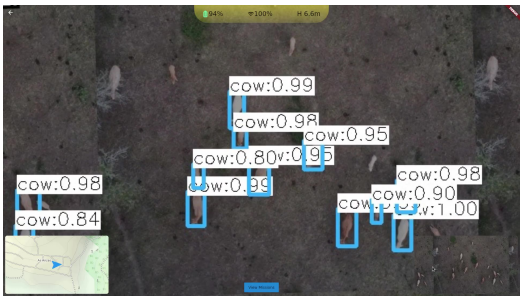
Resultados

El tiempo de procesado resultó ser el mismo tanto con la versión del *dataset* de COCO, como con la versión entrenada por nosotros. Los datos recopilados se muestran en la tabla 5.2.

(a) Ejemplo 1.



(b) Ejemplo 3.



(c) Ejemplo 4.

Figura 5.4: Varios ejemplos del procesamiento de las imágenes del dron.

El valor medio del tiempo de procesado es de 27.15 ± 0.18 ms (0.7%). Es un tiempo muy estable ya que el rango está entre 26 y 28 ms, es decir, sólo 2 ms de margen (cerca de un 7%). Nótese que procesar los frames por debajo de 33 ms permite procesar un video a 30 fps en tiempo real sin descartar ningún frame.

En la figura 5.4 se pueden ver algunos ejemplos del procesamiento de vídeo.

5.2.3 Pruebas de aterrizaje automático

Objetivo

Determinar la robustez del método de aterrizaje y medir la distancia desde la cámara del dron y el centro de la posición de aterrizaje.

Tabla 5.3: Resultados de las pruebas de aterrizaje automático del dron (más detalles en el texto).

			Posición inicial			
Video	**Prueba**	**Error (cm)**	**Altura (m)**	**Distancia (pixel)**	**Tiempo (s)**	**Video (m:s)**
1-7	1	8.0	6.2	287	29	00:08
	2	10.0	12.2	322	57	02:27
	3	8.5	7.0	440	30	04:38
	4	16.0	4.0	156	29	07:14
	5	10.0	11.4	325	56	09:05
	6	11.0	10.4	437	44	11:28
	7	18.5	12.5	264	47	13:07
8-13	8	14.5	7.9	238	28	00:48
	9	10.0	6.4	411	30	03:01
	10	9.0	6.4	222	24	04:43
	11	6.0	4.9	302	21	06:28
	12	22.0	4.2	378	27	08:13
	13	22.0	3.9	147	19	09:54
14-20	14	11.0	5.6	313	30	00:12
	15	15.0	7.7	149	32	01:50
	16	27.0	8.0	215	38	03:28
	17	4.5	5.0	294	30	05:10
	18	14.5	4.9	335	30	04:33
	19	14.0	5.5	305	23	07:56
	20	9.0	3.7	116	16	09:37
Media		13.33 $\pm$ 6.04				

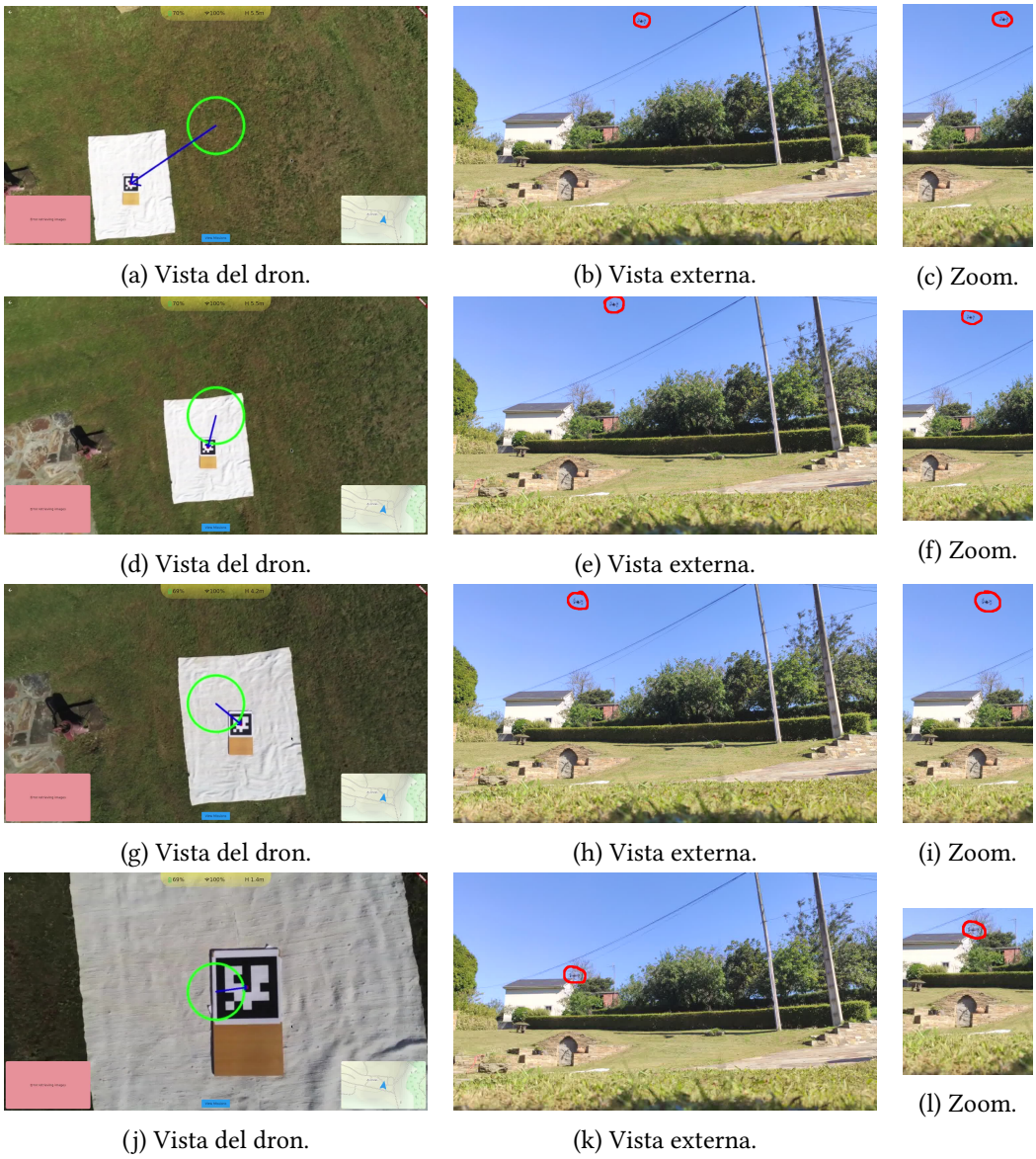
Metodología

Se comenzó probando el algoritmo 1, en donde se descubrió el fallo del balanceo que se comenta en Apartado 5.1.3 y se modificó probando los algoritmos 2 y 3, siendo el último el finalmente elegido para la solución final.

Un fallo en la conexión de vídeo entre el ordenador de control y el dron no reinicializaba la posición del punto de aterrizaje a 'desconocida', lo que provocaba que el dron continuara su movimiento en la dirección previa a la desconexión. Este fallo en la gestión de la pérdida de conexión fue causa del primer y único accidente que tuvo el dron en la realización de este proyecto. Para solventarlo se implementó un supervisor que está permanentemente controlando el correcto funcionamiento del sistema de recepción de vídeo, tratando de reiniciar la conexión en caso de pérdida de la misma.

Resultados

Los resultados de las pruebas de aterrizaje con el algoritmo 3 pueden ser consultados en la tabla 5.3. Los videos que se corresponden con esas pruebas están disponibles en el DVD y en el canal de *YouTube* en [33, 34, 35]. El error es la distancia entre la perpendicular de la cámara del dron y el centro del marcador Aruco que indica el punto de aterrizaje. Se muestran la altura como la distancia del dron al punto de recarga tanto en la primera como en la última



(a) Vista del dron. (b) Vista externa. (c) Zoom.

(d) Vista del dron. (e) Vista externa. (f) Zoom.

(g) Vista del dron. (h) Vista externa. (i) Zoom.

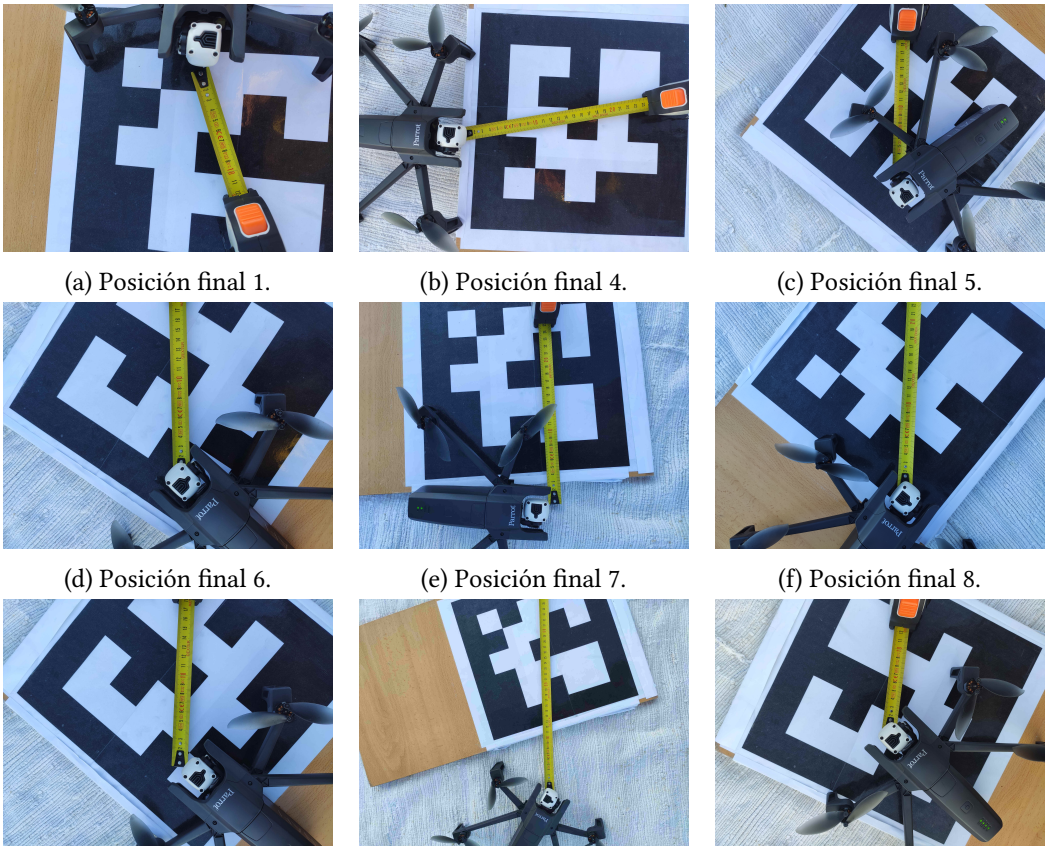
(j) Vista del dron. (k) Vista externa. (l) Zoom.

Figura 5.5: Imágenes del proceso de aterrizaje automático implementado. De izquierda a derecha: vistas desde el dron, foto tomada desde el exterior y ampliación de ésta última.

detección. También el tiempo total entre ambas imágenes.

Los experimentos se ha realizado en varias "tandas", es decir, con la batería del dron totalmente cargada, se le pedía a éste realizar una misión de seguir una ruta corta (rápida de ejecutar). Al finalizar cada ruta se repetía la misión hasta agotar la batería. Cada tanda permitía realizar hasta 7 misiones, con otros tantos aterrizajes, todos ellos realizados de forma totalmente automática.

En la figura 5.5 se pueden ver algunas imágenes representativas del proceso de aproxima-



(a) Posición final 1. (b) Posición final 4. (c) Posición final 5.

(d) Posición final 6. (e) Posición final 7. (f) Posición final 8.

(g) Posición final 9. (h) Posición final 16 (la peor). (i) Posición final 17.

Figura 5.6: Imágenes de la posición final del dron después de algunos aterrizajes automáticos.

ción durante algunos aterrizajes automáticos. En las imágenes se puede ver la altura a la que está el dron en cada momento y también la distancia y orientación (flecha azul en la imagen procesada) del punto de aterrizaje cuando se detecta (círculo verde en la imagen). En la misma figura se muestran también la vista externa del proceso.

Todas las imágenes mostradas en esta memoria se han extraído de los vídeos que se incluyen en el DVD adjunto a la copia impresa de este trabajo. También se pueden ver en un canal de Youtube.

Cabe señalar que después de tocar el suelo a veces el *firmware* del dron lo hace despegar de nuevo. Este fenómeno ocurre cuando el dron detecta que su posición final no es lo "suficientemente estable". Por ejemplo, cuando una pata está más alta que el resto (porque "pisa" la tabla donde se ha colocado el marcador que señala el punto de aterrizaje). Este fenómeno es bastante infrecuente (ocurre menos de un 10%) y se podría solucionar haciendo más plana toda la superficie de aterrizaje.

Otro fenómeno relevante es el viento en la zona de aterrizaje que, unidos a la latencia

medida en la sección anterior, hace más inestable el algoritmo diseñado. Por desgracia, no disponíamos de aparatos para medir la velocidad del viento para establecer de forma cuantitativa cómo afecta el viento al error y al tiempo de aterrizaje.

En la figura 5.6 se muestran algunas fotos sobre la posición final del dron después de varios aterrizajes representativos, justo mientras se mide el error cometido. El movimiento del dron previo a ceder el control al algoritmo de aterrizaje se hizo mediante los controles de teclado disponibles en la interfaz de linux, desplazándolo a varias posiciones aleatorias.

Como se puede observar, el error medio cometido es de 13.33 ± 6.04 cm (un error del 12%). Pero nunca superior a 27.0 cm. El mínimo error cometido es de 4.5 cm. Además de los aterrizajes mostrados en la tabla 5.3, se comprobó el funcionamiento del algoritmo en más de 50 vuelos. Cabe recordar que el tamaño del marcador Aruco es de 27 cm.

5.2.4 Dos puntos de aterrizaje

Objetivo

Demostrar que el dron puede aterrizar en varios lugares previamente designados con marcadores Aruco. De esta forma se podrá demostrar que el sistema puede aumentar de forma exponencial la autonomía del dron al no tener que volver siempre a un mismo punto de recarga.

Metodología

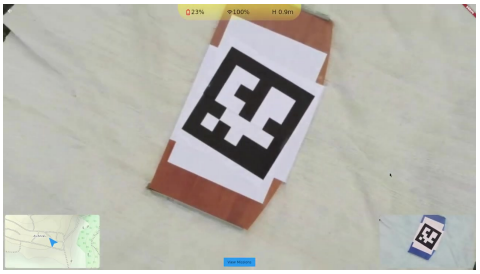
Se disponen dos puntos de aterrizaje relativamente cercano, para poder visualizarlos simultáneamente en algún momento del vuelo y se controla el dron para que despegue del primero y aterrice en segundo y después despegue del segundo y aterrice en el primero.

Resultados

En las figuras 5.7 y 5.8 se pueden ver algunas imágenes del dron mientras ejecuta la misión establecida. El video del cual se extrajeron las imágenes está accesible en el DVD y en [36]. En varias de las imágenes se pueden ver los dos marcadores Aruco que designan los dos puntos de recarga establecidos.

Esta prueba demuestra gráficamente la capacidad de nuestro sistema para mantener un dron en vuelo de forma indefinida, ya que se puede ir cambiando el punto de aterrizaje (y recarga) según las necesidades operativas de cada momento. Con el sistema propuesto, se pueden definir múltiples puntos de aterrizaje según la misión a realizar.

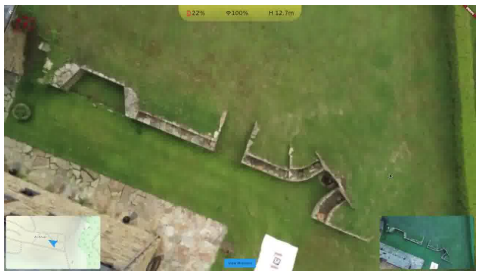
El límite de una misión lo marcaría las condiciones meteorológicas y el tiempo necesario para recargar la batería del dron o sustituirla por otra ya cargada.



(a) Despegue del primer punto de aterrizaje.

(b) Tomando altura.

(c) Altura operacional de la misión.



(d) Giro hacia el primer punto de aterrizaje.

(e) En el medio de la misión.

(f) Fin del proceso del desplazamiento.

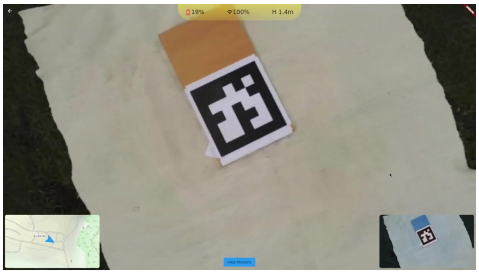
(g) Descenso al primer punto de aterrizaje.

(h) Aproximación al punto de aterrizaje.

(i) Inicio del aterrizaje.

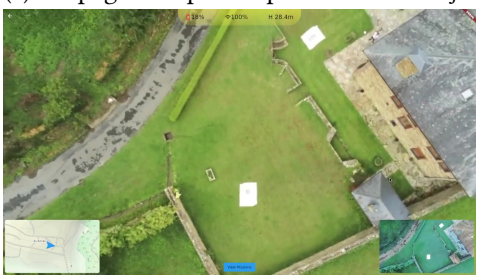
(j) Fase final del aterrizaje.

Figura 5.7: Secuencia de ida de imágenes con despegue y aterrizaje en puntos diferentes.



(a) Despegue del primer punto de aterrizaje.

(b) Tomando altura.



(c) Ya se ven los dos puntos de aterrizaje.

(d) Altura operacional de la misión.

(e) Fin del proceso del desplazamiento.

(f) Inicio de la fase de descenso.

(g) Acercamiento al 2o punto aterrizaje.

(h) Inicio del aterrizaje automático.

(i) Fase intermedia en el aterrizaje.

(j) Fase final del aterrizaje.

Figura 5.8: Secuencia de vuelta desde el segundo al primer punto de aterrizaje.

5.2.5 Prueba ejecución de misiones

Objetivo

Probar el funcionamiento del dron en un espacio abierto más grande al visto en las pruebas anteriores, mostrando como se planifica una misión y como se ejecuta. Probar que en el planificador de misiones se pueden especificar las coordenadas de los puntos y su altura y marcarlos como lugar de aterrizaje o despegue. Probar que desde la pantalla de control se ejecuta correctamente la misión creada.

Demostrar la capacidad del sistema para desplegar un dron en un tiempo muy reducido desde un punto móvil como es un remolque, realizando un desplazamiento con este desde el lugar de despegue al lugar de recogida del dron.

Metodología

Con la base de datos corriendo y, al menos la API HTTP arrancada, nos conectamos al planificador de misiones de la interfaz. Desde ella creamos una misión con un punto de despegue, varios puntos de destino y un punto final para el aterrizaje. Una vez tenemos planificada la misión pasamos a la segunda parte de la prueba, la ejecución de la misma.

Con el dron encendido, conectamos nuestra estación de control al mismo y, posteriormente accedemos a la pantalla de control del dron. Desde ella probamos que al indicar que se ejecute una misión, esta se ejecuta según lo indicado en el planificador.

Resultados

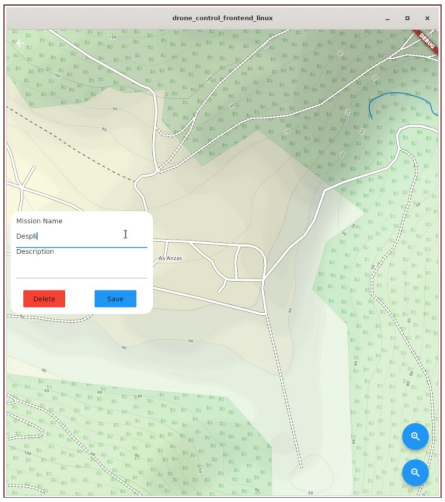
En las figuras 5.9 y 5.10 se puede ver el proceso seguido para crear una misión. En la figura 5.11 se pueden ver una secuencia de imágenes de la ejecución de la misión.

Como dato curioso, tanto el despegue como el aterrizaje tienen lugar desde un remolque que se ha desplazado desde el primer punto al otro. Como se puede observar, el remolque es un espacio muy acotado y sin ningún margen de error. Aunque también es verdad que es bastante más grande que las dimensiones del dron.

Sin embargo, hay que remarcar que el dron no ha seguido al remolque mientras ejecutaba su misión. Más aún, el remolque se encuentra parado durante el proceso de aterrizaje automático. Simplemente, se encontraba en el punto de aterrizaje definido en la misión (y conocido por el operador de antemano).

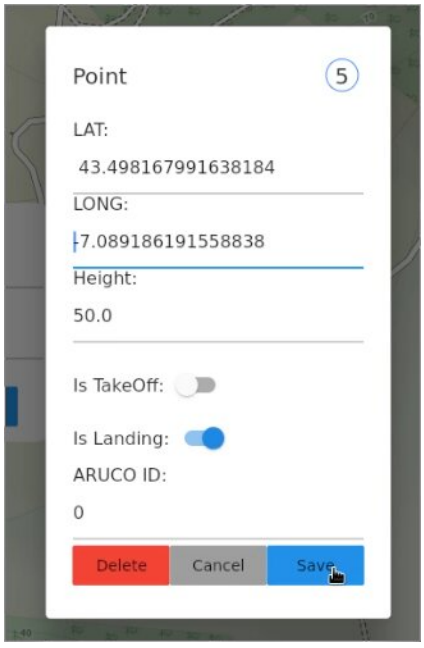
Esta prueba demuestra que, si se conocen uno o varios puntos GPS de destino, se pueden colocar en ellos un punto de aterrizaje (fijos o móviles) de forma que el drone pueda aterrizar en él de forma segura y aumentar de forma indefinida su autonomía.

Un último elemento a destacar en las últimas imágenes de la figura 5.11 es lo que cerca que pasa el dron del edificio en donde está aparcado la guía del remoque. Y lo seguro que se

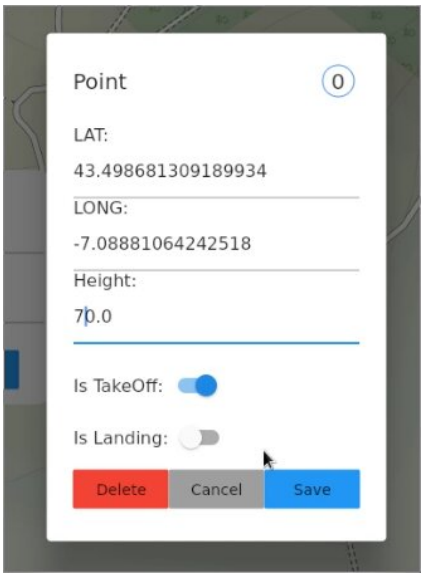


(a) Primeros puntos.

(b) Todos los puntos de la misión.



(c) Edición del punto de despegue.



(d) Edición del punto de aterrizaje.

Figura 5.9: Proceso de creación de una misión.

mueve el dron durante el acercamiento al punto de aterrizaje. Esta mision puede ser vista en video en [37].

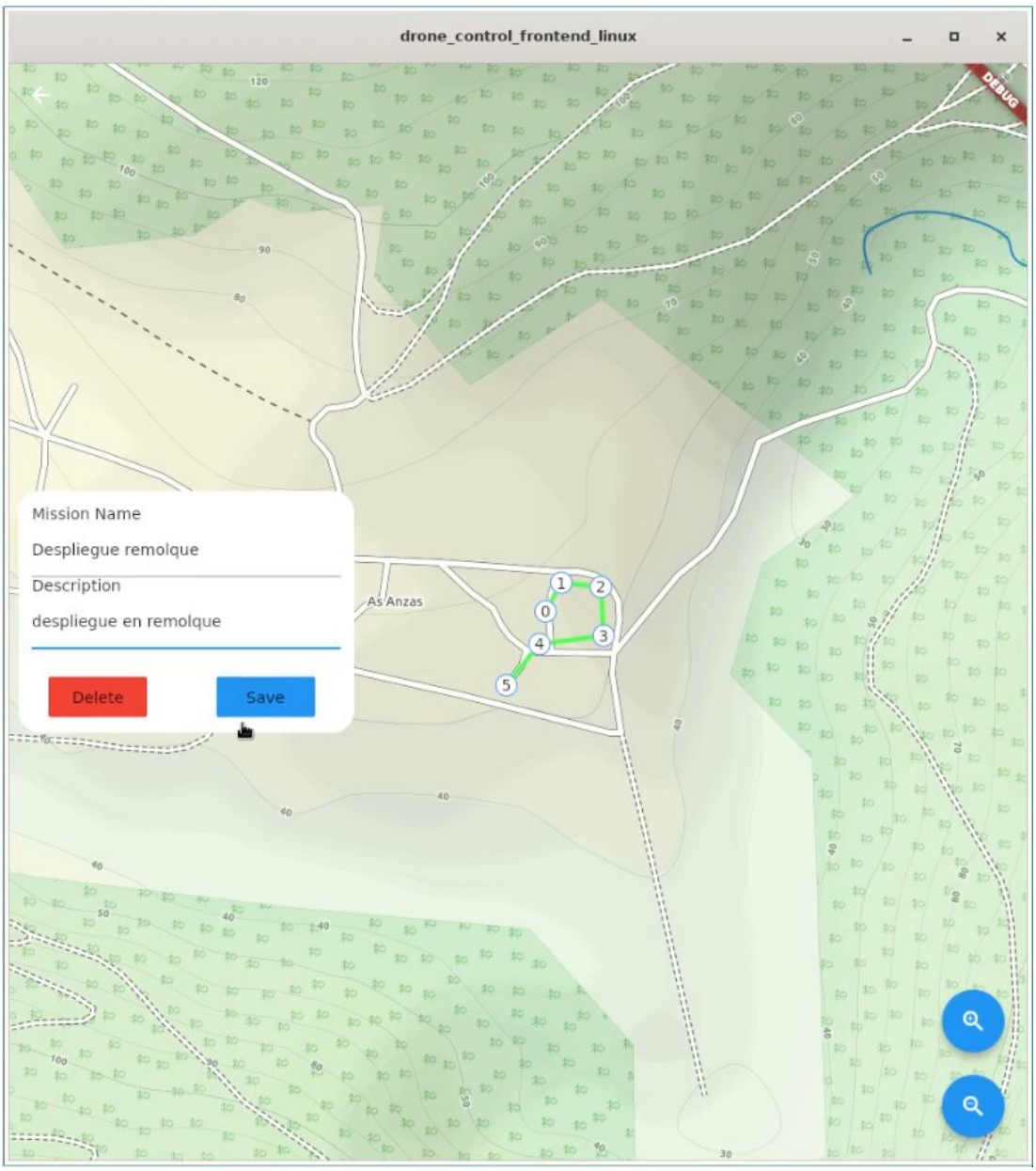


Figura 5.10: Vista del proceso de creación de la misión desde linux.

(a) Despegue (modo mapa). (b) Subiendo (modo mapa). (c) Llegada al punto 0.

(d) Orientado hacia el punto 1. (e) Llegada al punto 1. (f) Orientado hacia el punto 2.

(g) Llegada al punto 2. (h) Orientado hacia el punto 3. (i) Llegada al punto 3.

(j) Orientado hacia el punto 4. (k) Llegada al punto 4. (l) Orientado hacia el punto 4.

(m) Llegada al punto 5. (n) Orientado y descendiendo. (o) Detección punto aterrizaje.

(p) Bajada controlada. (q) Fase de acercamiento. (r) Fase final del aterrizaje.

Figura 5.11: Imágenes de una misión de larga duración con despliegue desde remolque.

Capítulo 6

Solución desarrollada

Como se indica en el apartado 4.2 el software desarrollado dispone de versiones en Linux, Android y Wear OS. A continuación se mostrará la estructura general y después las pantallas de las que dispone nuestra aplicación en los tres sistemas operativos.

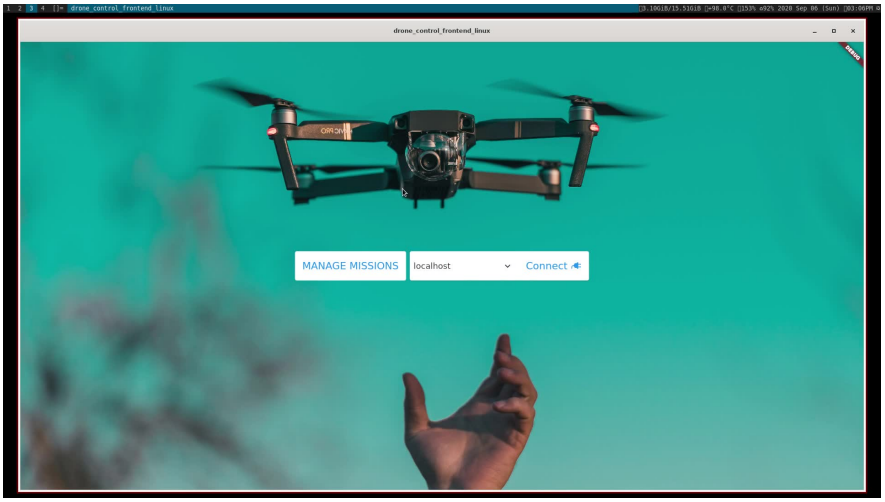
6.1 Elementos generales

La parte visible de la solución desarrollada está formada por dos grandes módulos y sigue el esquema mostrado en la figura 4.4:

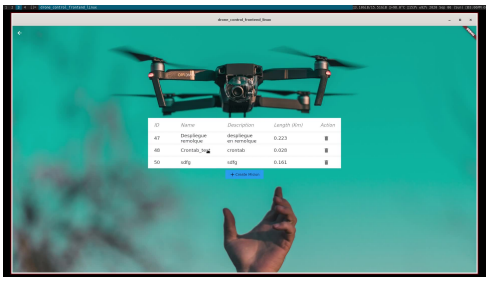
- **Creación de misiones**: Desde esta vista se permite al usuario crear nuevas misiones para ejecutar posteriormente con un dron. La personalización de la ruta permite añadir varios puntos de aterrizaje y despegue, además de los puntos por los que debe pasar el dron. A estos puntos se les puede editar altura y si son también puntos de aterrizaje o despegue.

- **Ejecución de misiones**: Permite visualizar la imagen en tiempo real del dron, así como la procesada y un mapa en el que se muestra la posición y dirección del mismo. En la parte superior se muestra la información que consideramos más relevante de cara a un usuario medio, incluyendo en esta vista la batería, calidad de la conexión con el dron y altura, una vez más en tiempo real.

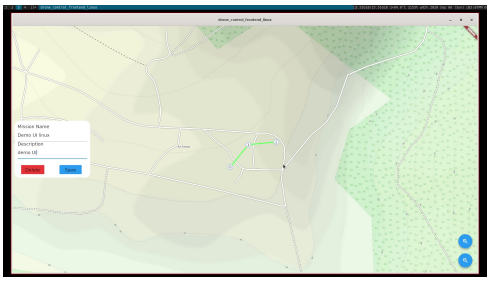
Gracias a la variedad de dispositivos desde los cuales se puede acceder al sistema, se consigue que el usuario final tenga disponible la visualización de lo que está sucediendo en todo momento con su dron, esta característica de nuestra solución se traduce en una ventaja competitiva con respecto a soluciones anteriormente vistas en el trabajo relacionado (apartado 2.1).



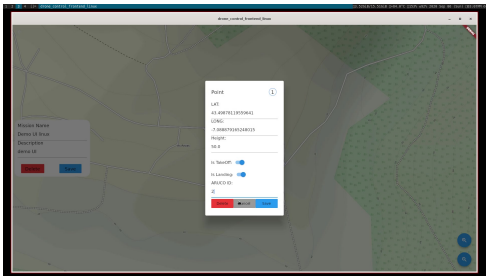
(a) Pantalla principal.



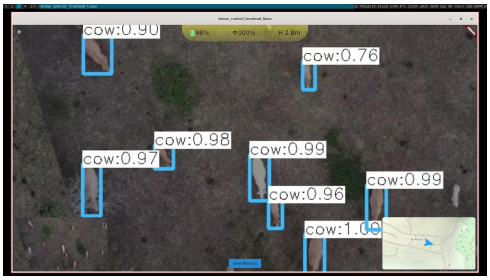
(b) Pantalla de visualización de misiones.



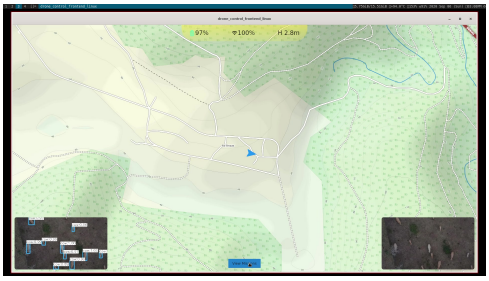
(c) Pantalla de creación de misiones.



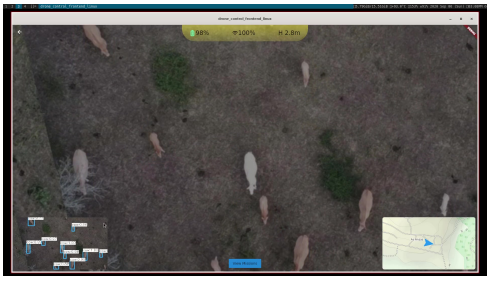
(d) Pantalla de creación de misiones.



(e) Pantalla de creación de misiones.

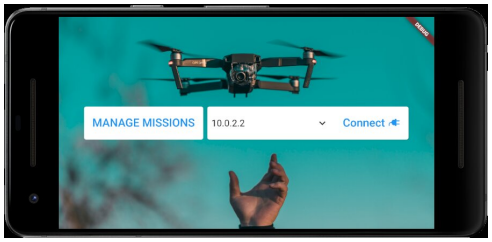


(f) Pantalla de control centrada en el mapa.

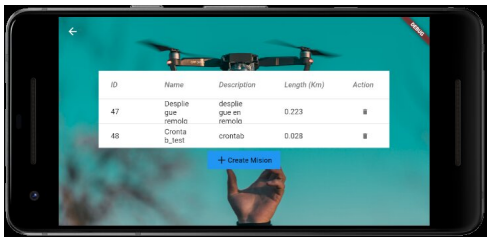


(g) Pantalla de control centrada en la cámara.

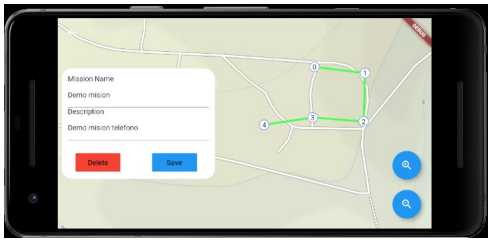
Figura 6.1: Versión de Linux del software desarrollado.



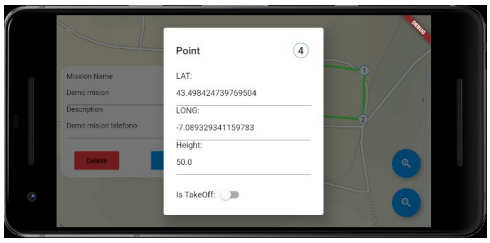
(a) Pantalla principal.



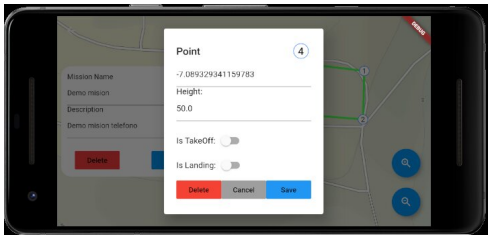
(b) Pantalla de visualización de misiones.



(c) Pantalla de creación de misiones.



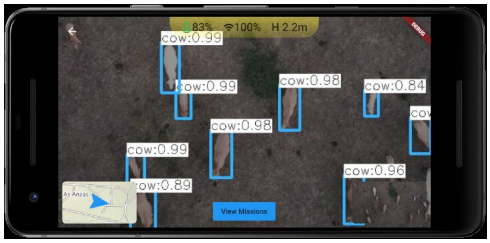
(d) Pantalla de creación de misiones.



(e) Pantalla de creación de misiones.

(f) Pantalla de control centrada en el mapa.

(g) Pantalla de control centrada en la cámara.



(h) Pantalla de control centrada en detector.

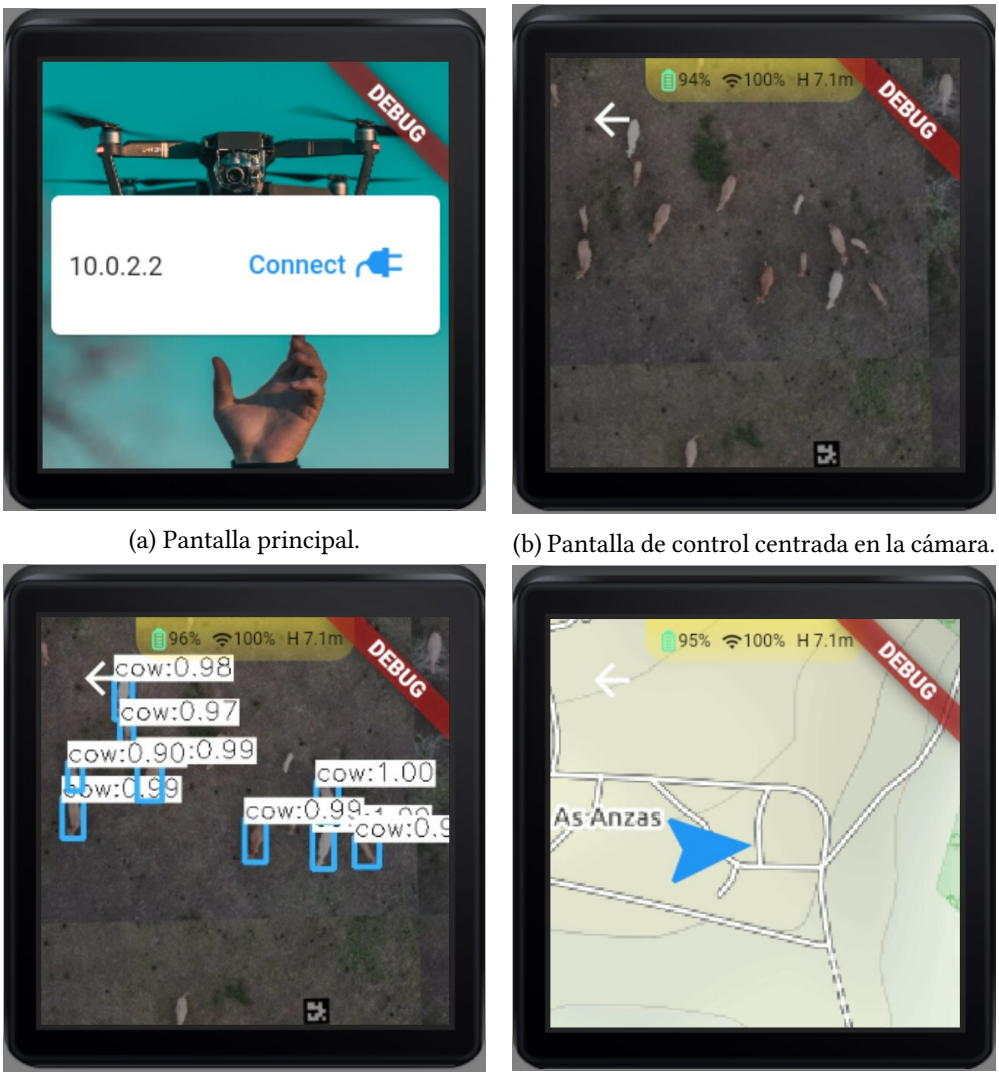
Figura 6.2: Versión de Android del software desarrollado.

6.2 Linux

La interfaz de Linux dispone de toda la funcionalidad del sistema, control por teclado inclusive. Las pantallas de las que dispone la aplicación se muestran en la figura 6.1.

6.3 Android

La interfaz es accesible desde Android (figura 6.2), con las mismas pantallas que en Linux pero sin la opción de control por teclado. Esta versión se puede ver en funcionamiento en el



(a) Pantalla principal.

(b) Pantalla de control centrada en la cámara.

(c) Pantalla de control centrada en el detector.

(d) Pantalla de control centrada en el mapa.

Figura 6.3: Versión de *smartwatch* del software desarrollado.

vídeo mostrado en [38].

6.4 SmartWatch

En esta versión, si tocamos sobre la pantalla de control, la vista rotará entre el mapa, el detector y la imagen directa de la cámara, tal y como se puede ver en la figura 6.3.

Capítulo 7

Conclusiones y trabajo futuro

Este último capítulo comenzará listando lo aprendido durante el desarrollo de este proyecto, seguirá con una comparativa con los objetivos que nos habíamos marcado y finalizará mencionando una serie de modificaciones o añadidos que nos gustaría hacer al proyecto en un futuro.

7.1 Lecciones aprendidas

Para la realización de este proyecto se buscó información en diferentes ámbitos, a continuación se puede ver una lista de las cosas aprendidas que consideramos más importantes.

1. **Linux**: Consolidación y aprendizaje de comandos de red (para conexión con el dron y comprobación de puertos ocupados).
2. **Protocolos**: Aprendizaje de protocolos de comunicación no vistos en la carrera (MQTT y WebSocket).
3. **opencv**: Aprendizaje en general de la librería de visión por computador, tanto en python como en c++.
4. **Flutter**: Adquisición de conocimientos en el ámbito del desarrollo de interfaces gráficas para linux con este framework, y porting del código a Android y Wear OS.
5. **Streaming de video**: Conocimientos en el formato de los frames MJPEG para retransmisión y recepción de los mismo.
6. **Patrones de diseño**: Aprendizaje del patrón BLOC para mantener el estado en aplicaciones flutter.
7. **Base de datos espacial**: Aprendizaje del funcionamiento de postgresql y postgis para el almacenamiento de datos espaciales.

8. **Pygame**: Uso de pygame para realizar un prototipado rápido para pruebas de un proyecto relacionado con la robótica.

9. **Cron**: Como usar crontab para la programación de tareas en linux.

10. **Visión artificial**: Se aprendió que los colores cambian según la luz incidente, dificultando la detección de colores en entornos reales, aún en HSV. Este fenómeno es especialmente molesto en superficies brillantes y se reduce en el caso de superficies mate.

11. **ffmpeg**: uso de ffmpeg para extracción de *frames*, recorte y compresión de videos.

7.2 Conclusiones

En esta sección compararemos los objetivos propuestos 1.2 con lo que finalmente se ha logrado en el proyecto.

El primer objetivo 1 consistía en diseñar un sistema capaz de crear misiones. De acuerdo con lo expuesto en la seción 5.2.5, hemos conseguido desarrollar un sistema con el cual un operador con una mínima formación es capaz de crear y editar misiones de una forma sencilla y muy intuitiva, por lo que damos este objetivo por cumplido.

El segundo objetivo 2 se centraba en ejecutar las misiones durante, virtualmente, largos períodos de tiempo. Tal y como se ha ilustrado en el apartado de pruebas (sección 5.2), consideramos que también hemos cumplido este objetivo con creces, ya que el dron puede despegar y aterrizar múltiples veces y en diferente puntos. Esta característica es fundamental para aumentar de forma virtualmente ilimitado el alcance del dron. Entendemos que la misión sólo estará limitada por las condiciones meteorológicas y el tiempo de recarga (o sustitución) de sus baterías

El tercer objetivo 2 consistía en el desarrollo de un detector que identificara el punto de aterrizaje en diferentes condiciones de iluminación y de visibilizad. Según lo visto en la sección 5.1.1, sí hemos conseguido realizar un detector basado en marcadores visuales Aruco capaz de ser resistente a oclusiones, pero aún no es válido en todas las condiciones de luz. Aún así, con una iluminación adecuada, el detector funciona con una tasa de acierto muy alta. La conclusión es que este objetivo no lo podemos considerar alcanzado en su totalidad.

El cuarto y último 4 objetivo determinaba que la realización de las pruebas se haría en un simulador y, si se pudiera, con un dron físico. Hemos podido adquirir un dron Parrot Anafi y lo hemos usado de forma extensiva en las pruebas. El modo de proceder siempre ha sido testar primero los algoritmos en el simulador, y posteriormente realizar las pruebas en el mundo real. Durante estas pruebas siempre han surgido problemas e inconvenientes no previstos ni modelables en el simulador. Hemos logrado solucionar con éxito todos estos imponderables, por lo que consideramos que este objetivo ha sido alcanzado por completo.

A mayores de los objetivos previstos en el inicio del proyecto, la solución desarrollada permite la integración con un procesador de imagen externo y la visualización en una interfaz amigable desde tres sistemas operativos diferentes: Linux, Android, o Wear OS, con la funcionalidad adaptada a cada una de ellos. También se pueden ejecutar misiones de forma automática, repetitiva y pre-programada, usando un *cronjob*.

Además, destacamos que dentro en nuestro sistema se pueden integrar teóricamente múltiples detectores que se pueden ejecutar en remoto, desde el ordenador de control de misiones, procesando el vídeo que se retransmite desde el dron.

Por último, conviene reseñar que nuestro sistema ha logrado aterrizar con éxito en espacios reducidos (p.e. un remolque) que se pueden desplazar fácilmente. Las únicas restricciones son conocer con precisión las coordenadas GPS del punto de aterrizaje y permanecer estático durante el aterrizaje.

El error medio cometido al aterrizar el dron en las 20 pruebas que se ha medido ha sido de 13.35 ± 6.04 cm, a pesar de tener una latencia en la retransmisión del vídeo de 235.4 ± 6.8 ms y un tiempo de procesado de imágenes para un detector tipo YOLO de 27.15 ± 0.18 ms. El máximo error cometido ha sido de 27 cm, curiosamente el tamaño del marcador visual Aruco empleado en las pruebas. Sin embargo, el número de aterrizajes (y por tanto de misiones) completadas con éxito supera claramente los 50.

7.3 Trabajo futuro

En un futuro nos gustaría desarrollar las siguientes:

- Implementar el servidor multimedia para mejorar la utilización de recursos en la estación de control.
- Añadir las opciones de ajuste del ángulo de la cámara y/o dirección del dron en cada punto de la misión, para hacerlas más flexibles.
- Securizar el control del dron.
- Cesión del control del dron a una segunda estación de control. Sería sencillo de realizar disponiendo de los recursos necesarios, y permitiría disponer de distintos puntos de recarga o sustitución de las baterías, extendiendo consecuentemente el rango de actuación del dron.
- Añadir el comando "COVER" al subsistema de ejecución de misiones Apartado 5.1.4, para poder cubrir una superficie sin necesidad de que el operador tenga que especificar manualmente todo el recorrido (puntos GPS) para realizar dicho "barrido".

- Modificar el esquema de base de datos para representar el ratio de descarga de distintos modelos de dron y representar mejor sus características.
- Integrar con otros entornos de desarrollo y de control de drones. Especialmente con ROS, que "de facto" es el estándar mundial en robótica.
- Conexión con una estación meteorológica local o remota, para comprobar si el vuelo es viable: velocidad del viento, lluvia, luz, etc.
- Mejorar la detección del punto de aterrizaje e incorporar elementos para hacerla más robusta (p.e. iluminación). También probar el aterrizaje automático en condiciones más adversas, e incluso relajar la restricción de estar estático.

Apéndices

Apéndice A

Contenido del DVD

En el DVD adjunto a esta memoria se encuentran todos los vídeos del canal de YouTube referenciamos en la misma, el fichero PDF y todo el código fuentes del proyecto Latex (incluídas todas las imágenes).

Lista de acrónimos

BLOC Micro Air Vehicle Link. 32, 33, 61

GNSS Global Navigation Satellite System.. 1

GPS Global Positioning System.. 15, 20, 29, 40, 63

HSV Hue Saturation Value.. 35

HTTP Hypertext Transfer Protocol.. 26

MAVLink Micro Air Vehicle Link. 3

MJPEG Motion JPEG.. 36, 40, 61

MQTT Message Queuing Telemetry Transport. 26, 61

SDK Software Development Kit.. 13, 20

YOLO You only look once.. vii, 6, 45

Glosario

Broker MQTT Servidor central que gestiona el modelo publicador/suscriptor.. 9, 11, 29

Smartwatch Reloj inteligente.. 31

Wear OS Sistema operativo para dispositivos corporales creado por Google.. v, 2, 17, 21, 25, 26, 29, 31, 57, 61, 63

Bibliografía

[1] D. G. Pulpeiro. Youtube channel with missions and more. Youtube. [Online]. Available: https://www.youtube.com/channel/UCukI-I0Qe8QiM65MIDZq3CQ/videos

[2] "Página web de paparazzi," 2020, (consultado el 09/08/2020). [En línea]. Disponible en: https://wiki.paparazziuav.org/wiki/Main_Page

[3] "Página web de qgroundcontrol," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: http://qgroundcontrol.com/

[4] "Página web de skyeyetech," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.azurdrones.com/product/skeyetech/#

[5] "Blog de parrot," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://blog.parrot.com/2020/07/08/hoverseen-partnership/

[6] "Página web debian," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.debian.org/index.es.html

[7] "Página web de dwm," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://dwm.suckless.org/

[8] "Página web de neovim," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://neovim.io/

[9] "Página web de vim," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.vim.org/

[10] "Github de tmux," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://github.com/tmux/tmux/wiki

[11] "Página web de vscode," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://code.visualstudio.com/

[12] "Página web de parrot sphinx," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://developer.parrot.com/docs/sphinx/whatissphinx.html

[13] "Página web de python," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.python.org/

[14] "Página web de c++ en wikipedia," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://en.wikipedia.org/wiki/C%2B%2B

[15] "Página web de dart," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://dart.dev/

[16] "Documentación de aruco," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit

[17] "Página web de parrot olympe," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://developer.parrot.com/docs/olympe/overview.html

[18] "Página web de flask," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://flask.palletsprojects.com/en/1.1.x/

[19] "Github de websocketserver," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://github.com/Pithikos/python-websocket-server

[20] "Página web de psycopg2," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.psycopg.org/

[21] "Github de flutter_map," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://github.com/fleaflet/flutter_map

[22] "Github de flutter_bloc," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://github.com/felangel/bloc/tree/master/packages/flutter_bloc

[23] "Github de opencv," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://github.com/opencv/opencv

[24] "Página web de postgres," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.postgresql.org/

[25] "Página web de postgis," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://postgis.net/

[26] "Página web de mosquitto," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://mosquitto.org/

[27] "Página web del manifiesto Ágil," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://agilemanifesto.org/

[28] "Página web de los principios del manifiesto Ágil," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: http://agilemanifesto.org/principles.html

[29] "Página web de pygame," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://www.pygame.org/news

[30] D. G. Pulpeiro. Scheduled deployment of a drone using crontab. Youtube. [En línea]. Disponible en: https://youtu.be/f9DGZKh6kEI

[31] ——. Latency test of drone's camera. Youtube. [En línea]. Disponible en: https://youtu.be/VeTU38YlzQg

[32] "Página web del dataset coco," 2020, (consultado el 10/08/2020). [En línea]. Disponible en: https://cocodataset.org/#home

[33] D. G. Pulpeiro. Precission landing algorithm tests 1-7. Youtube. [En línea]. Disponible en: https://youtu.be/SsFiiY9sgbk

[34] ——. Precission landing algorithm tests 8-13. Youtube. [En línea]. Disponible en: https://youtu.be/Y8vuQIhSZBE

[35] ——. Precission landing algorithm tests 14-20. Youtube. [En línea]. Disponible en: https://youtu.be/DAcR1UA9nEY

[36] ——. Drone autonomous mision with two landing positions. Youtube. [En línea]. Disponible en: https://youtu.be/Ffu91Q3Qeyc

[37] ——. Autonomous deployment of a drone from a trailer. Youtube. [En línea]. Disponible en: https://youtu.be/oP-tp5gVUfg

[38] ——. Autonomous drone fligth and landing, mission started and visualized from android. Youtube. [En línea]. Disponible en: https://youtu.be/aefc2P89pbA